FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Novas Metodologias de Controlo de Versões de *Software*

Miguel Luís da Silva Rentes

Relatório de Dissertação

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Jaime Enrique Villate Matiz (Professor Auxiliar, FEUP)

Orientador na EFACEC: Miguel Ferreira Pereira Gomes (Mestre em Informática, FCUP)

26 de Março de 2009

Novas Metodologias de Controlo de Versões de *Software*

Miguel Luís da Silva Rentes

Relatório de Dissertação
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: António Augusto de Sousa (Professor Associado, FEUP)

Arguente: João Luís Ferreira Sobral (Professor Auxiliar, Universidade do Minho)
Vogal: Jaime Enrique Villate Matiz (Professor Auxiliar, FEUP)

26 de Março de 2009

Resumo

O desenvolvimento de um projecto de *software* complexo com vários milhares de linhas de código, diferentes plataformas para testar e com vários *developers* a trabalhar sobre esse mesmo código, constitui um exercício de engenharia de *software* notável.
Neste cenário cada vez mais comum, é necessário manter e gerir o *software* para cumprir todos os requisitos para os quais foi desenvolvido (os requisitos pretendidos pelo cliente) mas também para auxiliar o processo de desenvolvimento do *software* (na equipa de *software developers*, de *software architects* e *software managers*).
Surge naturalmente a questão de manter as versões do *software* desenvolvido. Isto significa não só guardar o histórico das alterações por que passa o *software* a cada novo desenvolvimento, mas também auxiliar a tarefa de desenvolver o *software*: permitindo a criação de ramos privados de desenvolvimento, regredir no código até uma versão passada, marcar uma determinada versão do *software* com uma etiqueta, e aferir rapidamente que alterações foram feitas em determinada versão do *software*, entre outras tarefas.
Estas e outras questões são respondidas actualmente através do uso de SCM's[1], cujos objectivos são o de tomar nota (guardar o histórico) e controlar (gerir as revisões) toda e qualquer alteração por que passa o *software*.
A escolha de um SCM que satisfaça os requisitos acima e melhore significativamente o processo de desenvolvimento de *software* é assim, uma questão pertinente e de grande importância. Neste contexto em que surge a necessidade de adopção de um SCM, é altamente motivador estudar os actuais SCM existentes e influenciar positivamente o ciclo de desenvolvimento de equipas de *software* através do seu uso.
Esta dissertação é um estudo sobre o actual modelo de controlo de versões de *software* em ambientes colaborativos de desenvolvimento. Tem particular ênfase na melhoria considerável do ciclo de produção de *software*, extendendo as funcionalidades de um SCM (neste caso, o SVN) recorrendo unicamente a ferramentas *open source*.

[1] *Software Configuration Management*

Abstract

The development of a complex software project with thousands of code lines, different platforms to test and several developers (or teams of developers) to work on that same code, is indeed a remarkable software engineering exercise.
In this scenario, which is actually very common, it is necessary to maintain and manage the source code to fullfil the requirements for which it was developed (requirements wanted by the customer) but also to help the software development process (in the software development, software architecture and software management teams).
One question pops up naturally: how to maintain the different versions of the software we want to develop. This means not only keeping the history of all the changes that goes trough the software, but also helping the task of developing the software: allowing the creation of separate private development branches, reverting the source code to a previous stable version, tagging a given revision in time as a new version, and checking quickly what changes were done at a given source code version.
This and other questions are actually answered by the use of SCM's[2], whose goals are to take note (keep the history) and control (manage revisions) of any changes that affects the developed software.
The choice of a SCM that fullfils the above requirements and significantly improves the development process cycle is thus, a very important and pertinent question. In this context, where it urges the need to adopt a SCM, it is exciting to study the existing SCM's and positively influence the software teams through its use.
This thesis is a study on the actual software version control model in colaborative development environments. Its enphasis is on the significant improvement that the current development cycle can achieve by extending an SCM set of funcionalities (in this particular case, SVN) using exclusively *open source* tools.

[2]Software Configuration Management

Agradecimentos

Gostaria de agradecer ao Professor Jaime Villate, orientador da FEUP, por toda a disponibilidade e ajuda prestada para a resolução de dúvidas que foram surgindo e pelos debates e conversas informais sobre o fantástico universo do *software* livre. Obrigado por me mostrar, mais uma vez, que vale a pena ser diferente!
Ao Miguel Gomes, responsável da instituição, por todo o apoio, pontos de vista e disponibilidade que sempre demonstrou para a resolução das dúvidas que foram surgindo. Por último, pela amizade que sempre demonstrou desde que fui recebido na EFACEC.
Ao Professor Augusto de Sousa pela paciência e amabilidade em responder atempadamente às dúvidas que foram surgindo ao longo da dissertação.
A todos os colegas de trabalho que proporcionaram conversas interessantes sobre os tópicos abordados na dissertação e aqueles momentos inesquecíveis de camaradagem; em especial, ao Filipe Marinho, Eduardo Ramalho, Paulo Aguiar, Paulo Santos, Pedro Quelhas, João Luís Pinto, Paulo Sousa, Márcio Gonçalves, José António Simões, Diogo Mendonça, Eunice Filipe, Carla Santa Bárbara, Sandra Rodrigues, Dora Cabral, Paulo Viegas, Alberto Rodrigues, Alexandra Martins, Hugo Martins, Vítor Santos, Carlos Pereira, José Carlos Fonseca e Rui Rocha. A todos o meu sentido obrigado!
À minha família, pais, irmão, e sobrinho a quem dedico esta dissertação, pelo amor e carinho.
Por último, à Ana, pela paciência, carinho, dedicação e amor.

Miguel Luís da Silva Rentes

Aos meus pais, Luís e Maria
Aos meus avós, Maurício e Ana

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

CVS	*Concurrent Versions System*
SVN	*Subversion*
SCM	*Source Code Management*
VCS	*Version Control System*
DVCS	*Distributed Version Control System*
XP	*Extreme Programming*
API	*Application Programming Interface*
MIME	*Multipurpose Internet Mail Extensions*
CRLF	*Carriage Return Line Feed*
LF	*Line Feed*
UTC	*Coordinated Universal Time*
ISO	*International Organization for Standardization*
URL	*Uniform Resource Locator*
WebDAV	*Web-based Distributed Authoring and Versioning*
SSL	*Secure Sockets Layer*
SSH	*Secure Shell*
DOS	*Disk Operating System*
MD5	*Message-Digest algorithm 5*
IDE	*Integrated Development Environment*
UPS	*Unninterruptible Power Supplies*
SCADA	*Supervisory Control And Data Aquisition*
LAN	*Local Area Network*
VPN	*Virtual Private Network*

Simplicity is the ultimate sophistication.

L. Da Vinci

1 Introdução

Este capítulo serve como introdução ao tema da dissertação, começando por descrever o contexto na empresa onde o ambiente da dissertação ocorreu, o problema tratado, os objectivos e o planeamento do trabalho da dissertação. Por fim, é feita a descrição do que será abordado nos capítulos que se seguem.

1.1 Apresentação do Grupo EFACEC

A EFACEC é o maior grupo industrial português no domínio da produção de equipamentos eléctricos, electrónicos e da logística industrial, sendo também o maior sistemista na área das instalações eléctricas.
Com mais de 100 anos de história, o Grupo EFACEC teve a sua origem na *Moderna*, empresa nascida em 1905.
Constituída em 1948, o Grupo EFACEC tem cerca de 3000 colaboradores, estando presente em mais de meia centena de países. As actividades do Grupo EFACEC abrangem a concepção e a produção de equipamentos para áreas tão diversas como a Energia (transformadores de grande potência, transformadores de distribuição, disjuntores, seccionadores, quadros de distribuição modulares), a Electrónica Industrial (sistemas de informação, de alimentação, de telecontrolo, de sinalização ferroviária e de telecomunicações), a

Automação e Robótica (logística industrial, sistemas robotizados), o Ambiente (tratamento do ar, da água, de efluentes domésticos e industriais), as Energias Renováveis, e a concepção e realização de Sistemas chave-na-mão para a indústria e serviços.
Mais concretamente, o portfólio de actividades da EFACEC, está organizado nas seguintes unidades de negócio (Figura 1.1. Fonte: [EFA09b]):

Figura 1.1: Estrutura Societária da EFACEC.

- Energia:
  - Transformadores;
  - Aparelhagem de Média e Alta Tensão;
  - Servicing de Energia;
- Engenharia e Serviços:
  - Engenharia;
  - Automação;
  - Manutenção;
  - Ambiente;
  - Renováveis;
- Transportes e Logística:
  - Transportes;
  - Logística.

Estas áreas de negócio sustentam uma abordagem cada vez mais Sistemista/Integradora, satisfazendo as necessidades actuais do mercado e rentabilizando as várias valências do Grupo.
O Grupo EFACEC possui 6 unidades fabris (Porto, Maia, Ovar, Póvoa de Varzim, Macau e Argentina) e dois centros de engenharia de sistemas (Porto e Lisboa). Os mercados além fronteiras estendem-se por todos os continentes, nos quais o Grupo EFACEC se faz representar por filiais e agências ou ainda pela constituição de *joint ventures* com empresas locais (Figura 1.2. Fonte: [EFA09c]).

Figura 1.2: Centros Fabris e Filiais da EFACEC no Mundo.

A aposta da EFACEC no mercado internacional (em seis regiões internacionais prioritárias: EUA, América Latina, Europa Central, Magrebe, África Austral e Espanha), bem como um forte investimento na Inovação e no Desenvolvimento de novas tecnologias, em articulação com as tecnologias de base, fazem com que a EFACEC tenha sabido penetrar favoravelmente no mercado, posicionando-a na linha da frente da indústria portuguesa e nos mercados internacionais (Figura 1.3. Fonte: [EFA09a]).
Estes factores são a base para o crescimento e o desenvolvimento sustentados do Grupo EFACEC, facturando anualmente aproximadamente 500 milhões de euros, exportando cerca de metade da sua produção.

1.1.1 EFACEC Engenharia, S.A.

A EFACEC Engenharia S.A. é uma empresa do Grupo EFACEC, constituída em 2007 a partir da anterior EFACEC Sistemas de Electrónica S.A., constituída em 1991 como empresa autónoma, tendo surgido como uma evolução natural da anterior Electrónica Industrial, criada em 1979 dentro da EFACEC Empresa Fabril de Máquinas Eléctricas, S.A.R.L..
A empresa desenvolve a sua actividade em diversas áreas de negócio que adoptam como base as Tecnologias de Informação e a Electrónica, fazendo uso intensivo de Sistemas de

Figura 1.3: EFACEC no Mundo.

Telecomunicações. A cada área de negócio corresponde, do ponto de vista de organização, uma unidade:

- **Sistemas de Energia, Automação e Telecontrolo**: esta unidade dedica-se à automação de estações e subestações (no que diz respeito à distribuição e transporte de electricidade), bem como a Sistemas de Gestão e Supervisão para redes eléctricas e de água;

- **Sistemas de Transporte**: a unidade de sistemas de transporte trata da sinalização dos transportes ferroviários, bem como das necessidades dos caminhos-de-ferro e estradas no que diz respeito à gestão e sistemas de suporte de informação pública;

- **Sistemas de Fornecimento de Energia**: a unidade de sistemas de fornecimento de energia desenvolve, comercializa e constrói UPS [1] e baterias, assim como rectificadores e conversores de energia eléctrica para aplicações especiais;

- **Soluções Integradas**: a unidade de soluções integradas fornece e instala infraestruturas para telecomunicações integradas para os operadores e para redes dedicadas nos sistemas dos caminhos-de-ferro e estradas. A empresa tem recursos para a instalação e serviços bem como para a produção de equipamentos completos de energia e controlo.

[1] *Unninterruptible Power Supplies*

1.2 Sistemas SCADA

Os primeiros sistemas SCADA[2], basicamente telemétricos, permitiam informar periodicamente o estado corrente do processo industrial, monitorizando sinais representativos de medidas e estados dos dispositivos, através de um painel de lâmpadas e indicadores, sem que houvesse qualquer interface aplicacional com o operador. Com a evolução tecnológica, os computadores assumiram um papel de gestão na recolha e tratamento de dados, permitindo a sua visualização num écran e a geração de comandos de programação para execução de funções de controlo complexas [Pin04].
Actualmente os sistemas SCADA utilizam tecnologias de computação e comunicação para automatizar a monitorização e controlo dos processos industriais, efectuando recolha de dados em ambientes complexos, dispersos geograficamente, e a respectiva apresentação de modo amigável para o utilizador, com recurso a interfaces gráficas. Este tipo de sistemas cobre um mercado cada vez mais vasto, podendo ser encontrados em diversas áreas, que vão desde a indústria electrónica até à indústria têxtil e revelam-se de crucial importância na estrutura de gestão das empresas, factor pelo qual deixaram de ser visto como meras ferramentas operacionais ou de engenharia, e passaram a ser encarados como uma importante fonte de informação. A melhoria do processo de monitorização e controlo, disponibilizando em tempo útil o estado actual do sistema, através de um conjunto de previsões, gráficos e relatórios, permite a tomada de decisões operacionais apropriadas, quer automaticamente, quer por iniciativa do operador.
Hoje em dia, este tipo de sistemas é intrinsecamente distribuído, com utilizadores e dispositivos dispersos, utilizando uma ou várias bases de dados e suportando desenvolvimentos incrementais e várias plataformas de computação (Linux [Wik09h], UNIX [Wik09r], Windows [Cor09e]). A tolerância a falhas e a redundância são também requisitos importantes.
Do ponto de vista do utilizador o sistema interage com um enorme volume de dados, e deverá ser fácil de operar, enquanto que, do ponto de vista do vendedor, o sistema deve poder ser costumizado de forma a responder a cada especificação do cliente.

1.3 Contexto/Enquadramento

Desenvolver um sistema SCADA é uma tarefa complexa, laboriosa e com várias especificidades técnicas. O processo de construção deste tipo de *software* tem de obedecer a critérios rigorosos por forma a assegurar que cumpre os requisitos deste tipo de *software* e a auxiliar os *developers* a programar um sistema tão complexo como um sistema SCADA.

[2] *Supervisory Control And Data Aquisition*

Neste contexto, o uso de um *software* de controlo de versões – SCM[3] – é a opção mais viável a tomar para controlar as alterações ao código-fonte e gerir todo o histórico do *software*.
A escolha de um SCM é uma questão delicada e importante no seio de uma equipa de *software*. É delicada pois vai introduzir mudanças no ciclo de trabalho de cada *developer* (para o bem comum de todos, espera-se) e é importante porque vai gerir todo o trabalho dessa mesma equipa de trabalho. É pois, nesta perspectiva, uma escolha difícil mas de grande relevância. É inclusivamente muitas vezes pouco consensual, pois introduzir novas formas de trabalho na equipa é intrusivo e toma sempre tempo para a adaptação; razões pelas quais travam a implementação de um ciclo de trabalho renovado junto da equipa. No entanto, se a escolha for a acertada (e com acertada, significa a escolha que maximiza os benefícios do controlo de versões do *software* e minimiza os danos colaterais), toda a equipa vai beneficiar deste facto e com a experiência melhorar a sua forma de trabalho para tirar uma maior eficiência do sistema de controlo de versões adoptado.
Numa equipa pequena de desenvolvimento (cerca de 30 pessoas distribuídas por 5-7 projectos que se interligam entre si), a decisão de escolher um SCM é naturalmente uma questão que terá impacto na forma de trabalho das equipas de desenvolvimento de *software* e de qualidade do *software*. É neste contexto que a investigação para esta dissertação nasce: qual a melhor forma de escolher um SCM, de o integrar nas equipas de desenvolvimento e de qualidade, e de melhorar o seu uso diário para uma maior produtividade no ciclo de produção de *software*.

1.4 Motivação e Objectivos

Influenciar a escolha de um SCM, melhorar o ciclo de desenvolvimento de *software* e aumentar a produtividade dentro da equipa de desenvolvimento ao automatizar determinadas tarefas (que serão detalhadas mais tarde), é uma actividade delicada mas altamente motivadora. Por isso mesmo, e porque o trabalho expresso na dissertação pode muito bem ser aplicado a outras equipas de departamentos diferentes na EFACEC (ou inclusivamente a outras empresas nacionais ou internacionais), os objectivos traçados foram delineados de modo a permitir um estudo alargado dos SCM's disponíveis hoje em dia, das ferramentas mais utilizadas com os SCM's de modo a tirar o máximo proveito dos mesmos, e da forma como os integrar no ciclo de trabalho das equipas de desenvolvimento e de testes de *software*.
Temos assim, como objectivos da dissertação:

- Detalhe do Estado-da-Arte dos SCM's adoptados hoje em dia, com especial ênfase no *software* livre SVN;

[3] Source Code Management

- Análise do Subversion como SCM e dos benefícios concretos da sua adaptação ao ambiente empresarial, entre os quais se realça a possibilidade de extender o SVN e construir um SCM de elevado valor;
- Aferição dos casos de uso do Subversion, com exemplos de uso em ambiente empresarial;
- Análise das ferramentas disponíveis hoje em dia para a integração do SVN no modelo de construção de *software* em ambientes de trabalho distintos;
- Caso de estudo EFACEC: análise do modelo de versionamento dos projectos de *software* na divisão de I&D da EFACEC;
- Programação de uma aplicação em Java que usa o Subversion (biblioteca SVNKit) para auxiliar o actual processo de criação/teste de *software* na divisão de I&D da EFACEC;
- Adaptação do SVN, das ferramentas disponíveis e da aplicação Java no modelo de desenvolvimento de *software* na divisão de I&D da EFACEC;
- Escrita de dois artigos: um de divulgação junto da comunidade online portuguesa de programadores P@P [aP09] (Portugal-a-Programar), que divulgue o controlo de versões de *software* e o Subversion numa óptica de utilização diária pouco complexa, os benefícios de uso de um *software* de controlo de versões e como instalar e configurar o Subversion; E um segundo artigo, numa revista especializada, que resuma a dissertação, os objectivos atingidos e as conclusões derivadas do trabalho desenvolvido ao longo da dissertação.

1.5 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 4 capítulos, ao longo dos quais se descreve o trabalho desenvolvido no decurso da dissertação e os conceitos fundamentais à sua compreensão. Estes são:

- Capítulo 2 – *Revisão Bibliográfica* – é descrito o estado-da-arte dos SCM's utilizados hoje em dia e o conjunto de ferramentas associadas aos SCM's mais usados;
- Capítulo 3 – *Desenvolvimento na EFACEC: um Caso de Estudo* – é analisado o processo de desenvolvimento de *software* no departamento de I&D da EFACEC, antes do estudo efectuado no âmbito da dissertação;
- Capítulo 4 – *Metodologias Propostas* – é a descrição das metodologias propostas para o uso do SCM Subversion e para a solução do problema apresentado no

capítulo 3, descrevendo as tecnologias a usar e as alterações necessárias para implementar as metodologias propostas;

- Capítulo 5 – *Conclusões e Trabalho Futuro* – é apresentado o cojunto de conclusões a retirar do estudo efectuado na dissertação e apresentado algumas propostas de melhorias e do rumo a seguir para o trabalho futuro neste tema.

No Anexo A encontra-se uma introdução ao SVN e à sua utilização, ao passo que no Anexo B é descrito como se instala e configura o SVN para ser utilizado com o servidor HTTP da Apache [Fou09a] e o sistema de *bug tracking* Trac [Sof09c]. Estes dois anexos deverão ser lidos para quem está a começar a usar o Subversion e necessita de rapidamente saber o que fazer para dar os primeiros passos na utilização deste sistema de controlo de versões.

If we really understand the problem, the answer will come out of it, because the answer is not separate from the problem.

J. Krishnamurti

2 Revisão Bibliográfica

O controlo de versões de *software* é actualmente um problema com elevada complexidade e uma área importante na construção de *software* numa perspectiva de engenharia de *software* [Som06].
Neste capítulo é apresentado o estado-da-arte dos SCM's e as principais ferramentas utilizadas para integrar mais facilmente os SCM's no ciclo de desenvolvimento de *software*.

2.1 Visão Geral

Os sistemas de controlo de versões – VCS[1] ou SCM[2] – permitem aos *developers* ou autores guardarem um histórico dos seus projectos (sejam de *software* ou não): obter facilmente uma versão antiga do projecto, manter vários ramos de desenvolvimento, fundir[3] várias correcções distintas numa única versão e poder aferir um conjunto alargado de informações respeitantes ao projecto que está a ser versionado .
Para resolver estes problemas, várias propostas foram criadas, umas comerciais outras de código aberto. Nas secções seguintes serão analisados os principais SCM utilizados hoje em dia relativamente às suas funcionalidades, filosofia de uso e ao modelo de gestão das alterações num dado projecto versionado.

[1] *Version Control System*
[2] *Source Code Management*
[3] Em Inglês, *merge*

2.2 Sistemas de Controlo de Versões

Um sistema de controlo de versões é o sistema responsável por gerir as inúmeras revisões pelas quais passa um determinado objecto de informação. Este objecto de informação pode ser o mais simples possível, como um único ficheiro de alguns *bytes* de tamanho, ou um projecto complexo de *software*, mantido por vários *developers*, com inúmeros ficheiros relacionados entre si para resultar numa aplicação com valor para o utilizador final.
Os sistemas de controlo de versões são aplicações em si, mas o controlo de revisões está actualmente integrado em vários tipos diferentes de *software* como IDE's[4], sistemas de gestão de conteúdos, processadores de texto e de cálculo, *wiki*'s, entre outros. Estes sistemas de controlo de versões estão a ter actualmente um enorme reconhecimento e uma importância vital para a organização de projectos de *software* cada vez mais complexos e com equipas de desenvolvimento distribuídas geograficamente.

2.2.1 Modelo de Gestão Centralizado

Os sistemas de controlo de revisões tradicionais adoptaram um modelo centralizado, onde todas as funções de controlo de revisões são realizadas por um só servidor central (Figura 2.1). Se dois *developers* tentam alterar um mesmo ficheiro ao mesmo tempo, sem qualquer tipo de método que proteja o acesso ao servidor, os dois *developers* podem re-escrever acidentalmente o código um do outro. Os sistemas de controlo de versões centralizados resolvem este problema com um de dois modelos de gestão possíveis: *locks* (ou seja, bloqueio temporário de um recurso para que apenas um *developer* o possa alterar e assim se proteger de um acesso concorrente a esse recurso) de ficheiros ou *merge*[5] das versões.

2.2.1.1 *Locks* de Ficheiros

O método mais simples de prevenir problemas de acesso conconrrente é bloquear ficheiros para que apenas um *developer* tenha acesso de escrita a esses ficheiros copiados do repositório central. Assim que um *developer* obtenha esse *lock*, fazendo um *check out*[6], outros *developers* podem ler esse ficheiro, mas nenhum pode alterar esse mesmo ficheiro até que o primeiro *developer* faça um *check in*[7] à versão alterada (ou cancele o *check out*).

[4] *Integrated Development Environment*

[5] Um *merge* é a fusão de código de diferentes *developers* numa mesma versão. Ao longo da dissertação adopta-se esta expressão em detrimento da palavra Portuguesa "fusão".

[6] Uma operação de *check out* de um ficheiro significa trazer do servidor para uma área de trabalho local uma cópia com permissão de escrita desse ficheiro.

[7] Ou seja, coloque o ficheiro alterado no servidor para que outros o possam escrever.

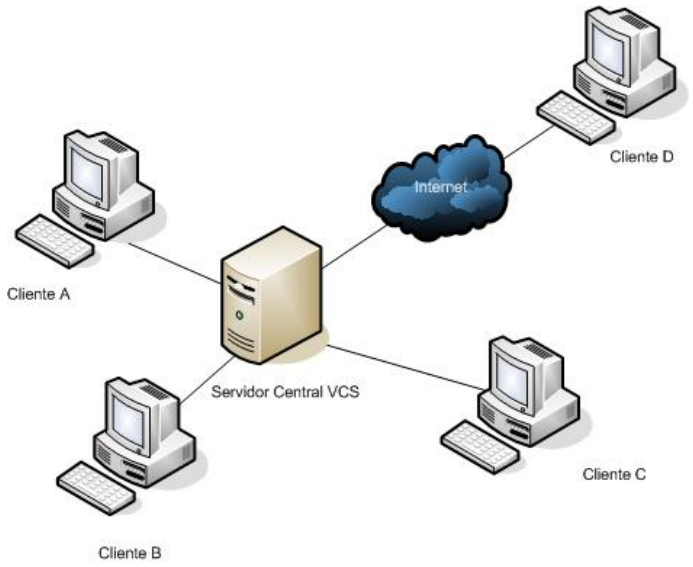

Figura 2.1: Modelo de gestão centralizado.

O *lock* de ficheiros tem vantagens e grandes desvantagens. Pode fornecer alguma protecção contra conflitos[8] em *merges*, que podem ser difíceis de resolver, quando um *developer* está a fazer muitas alterações em várias secções diferentes de um conjunto alargado de ficheiros. No entanto, se os ficheiros são ”abandonados”por um longo período de tempo pelo mesmo *developer* (ou quando este inevitavelmente vai de férias e se esquece que tem ficheiros *locked*), outros *developers* não podem trabalhar nesses ficheiros. Nesta situação é uma tentação circunscrever o sistema de controlo de versões e alterar os ficheiros localmente (fazendo em seguida um *check in* forçado), o que vai levantar novos problemas assim que o primeiro *developer* tente colocar as suas alterações no repositório.
É actualmente recomendado o uso de *locks* apenas em situações muito específicas e controladas. Por exemplo, na alteração de uma imagem (que é um ficheiro binário) faz todo o sentido fazer um *lock* desta imagem e apenas uma pessoa trabalhe exclusivamente sobre essa imagem. Quando esta pessoa terminar o trabalho e fizer *check in*, toda a equipa pode actualizar a sua cópia de trabalho e obter a imagem alterada. Isto não significa que não existam VCS que consigam computar diferenças entre as alterações de vários *developers* nos mesmo ficheiros binários. É sim mais fácil para nós humanos lidar com possíveis conflitos em ficheiros de texto do que em ficheiros binários. Daí que seja importante usar o *lock* de ficheiros apenas em caso muito específicos.

2.2.1.2 *Merge* de Alterações

A maioria dos VCS actualmente permite que todos os *developers* possam editar o mesmo ficheiro ao mesmo tempo. O primeiro *developer* que faça *check in* altera esse ficheiro

[8]Um conflito surge aquando de uma alteração numa mesma zona do código versionado. Tipicamente, na escrita nas mesmas linhas de código.

com sucesso no repositório central. Os outros *developers* ao tentar fazer *check in* têm obrigatoriamente que actualizar as suas cópias de trabalho locais e só então realizar um *check in*, que pode gerar um conflito, apenas no caso em que as mesmas zonas do ficheiro (ou conjunto de ficheiros) tenham sido escritas pelo *developer* que realizou o *check in* anterior.
Alguns VCS, como o Subversion [Col09b], possui o conceito de *edição reservada*, que fornece a possibilidade opcional de usar o *lock* de ficheiros para usar exclusivamente o acesso de escrita de um ficheiro, apesar de existir a funcionalidade de *merge* das alterações nos ficheiros.

2.2.2 Modelo de Gestão Distribuído

O modelo de gestão distribuído (DVCS) é uma inovação recente que fornece algumas vantagens sobre a tradicional abordagem aos sistemas de controlo de versões, e contém algumas características que o separam claramente deste último tipo de sistemas. No entanto, a linha que separa os sitemas distribuídos dos centralizados tem ficado um pouco tenue, uma vez que os DVCS's podem ser utilizados em modo "centralizado". Um sistema distribuído toma sempre uma abordagem P2P[9], em que em vez de existir um só repositório central no qual todos os utilizadores sincronizam as suas cópias locais de trabalho, cada cópia local de trabalho de cada *peer* é em si um repositório. A sincronização é feita através da troca de *patches* (ou seja, conjuntos de alterações) de *peer* para *peer* (Figura 2.2). Isto resulta em importantes diferenças relativamente aos sistemas centralizados:

1. Existem vários repositórios centrais, ao passo que na abordagem centralizada há apenas um;
2. O código presente nos diversos repositórios centrais é unificado baseando-se numa rede de confiança, ou seja, no mérito histórico das alterações ou na qualidade das alterações em si;
3. Os *Tenentes* são membros de um projecto que têm o poder de decidir dinâmicamente que ramos vão usar para fazer o *merge* das suas alterações;
4. O acesso à rede não é utilizado na maioria das operações;
5. As operações comuns como *commit's* (ou *check in*), ver o histórico, reverter as alterações feitas, etc., são extremamente rápidas, uma vez que não há a necessidade de comunicação com o servidor central. Ao invés, a comunicação é apenas necessária quando se colocam alterações (*push*) ou se trazem alterações (*pull*) entre *peers*;

[9] *Peer-to-peer*. Um *peer* corresponde a um participante na rede de *peers*.

6. Cada cópia local de trabalho é efectivamente uma cópia remota do código-base e do histórico de alterações, fornecendo uma protecção natural contra perdas de dados, porque existe replicação ao longo da rede P2P.

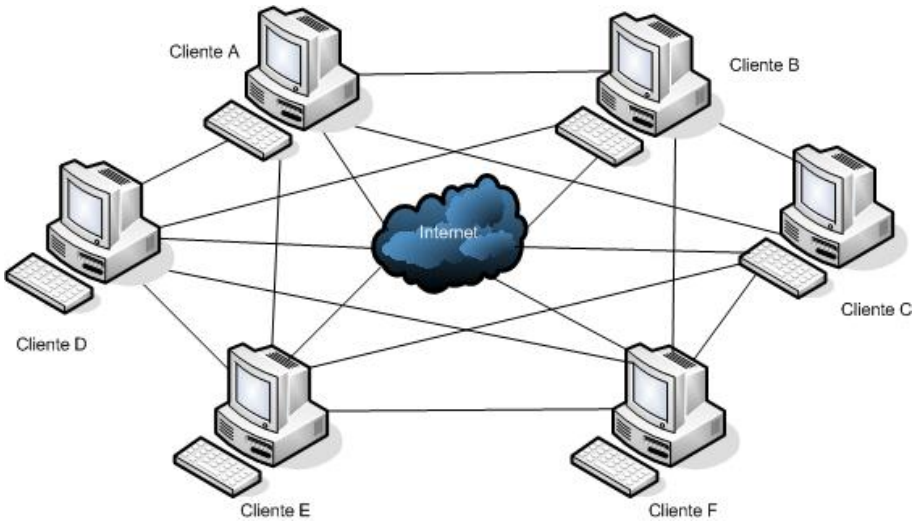

Figura 2.2: Modelo de gestão distribuído.

Como se pode ver, é uma filosofia de uso altamente distinta da abordagem tradicional de repositório de VCS centralizado.
Como vantagens, um sistema distribuído permite que um *developer* trabalhe efectivamente sem necessitar de um acesso à rede do repositório, o que também influencia a rapidez e *performance* das operações, pois estas não dependem do acesso à rede; Para além destas vantagens, permite mais facilmente as contribuições de qualquer *developer* uma vez que não há necessidade de pedir permissões aos responsáveis pelo projecto para começar a desenvolver correcções ou novas características nesse mesmo projecto. Este tipo de sistema permite também criar áreas privadas de desenvolvimento sem que ninguém se aperceba que se está a desenvolver num ramo privado do projecto: é uma autêntica *sand box* privada. No caso da abordagem centralizada, isto não é possível sem que se crie um ramo que é público para toda a equipa de desenvolvimento. Por fim, este tipo de sistema não depende de uma só máquina servidora, pelo que nunca vai existir qualquer tipo de problemas derivados de um *crash* de discos.
Como desvantagens face a um sistema centralizado, não é possível remover ou alterar o histórico apenas por uma pessoa. Isto pode ser uma limitação grave num ambiente empresarial, especialmente quando é necessário ter todo o controlo sobre o histórico e poder realizar qualquer tipo de operação sobre o mesmo. Outra desvantagem, e igualmente num ambiente empresarial, é a maior dificuldade em gerir os acessos ao conteúdo e a que componentes se podem aceder em determinado projecto versionado. Esta questão é facilmente resolvida numa abordagem centralizada, que fornece maior segurança informática

nos acessos. Por fim, os conceitos de utilização de um DVCS são mais difíceis de apreender por parte dos *developers*, uma vez que é necessário entender mais conceitos sobre a própria infra-estrutura que está inerente a um sistema distribuído.

2.3 Vantagens de um Sistema de Controlo de Versões

As funcionalidades básicas de qualquer sistema de controlo de versões são o de permitir anotar todas as alterações dos ficheiros ao longo do tempo e de unir as contribuições de vários programadores. Para suportar isto, o VCS tem de manter um histórico de todas as alterações que foram feitas ao longo do tempo por diferentes pessoas. Desta forma, é possível voltar atrás no código até uma revisão mais antiga, e ver que aspecto tinham os ficheiros nessa altura. Para além disso, o VCS tem de garantir sempre a correcta integração das alterações feitas no código.
A pergunta que naturalmente se coloca é a de que tipo de vantagens poderão advir da adopção de um SCM. Especialmente, para uma equipa pequena que benefícios se podem tirar de um bom VCS que valham a pena consumir tempo a aprender como usá-lo no dia-a-dia? Eis algumas das razões para usar sempre um VCS [Nag05]:

1. **Integridade dos dados** – Um bom VCS ajuda a proteger a integridade dos dados. Ao manter um histórico das revisões (que são os conjuntos atómicos de alterações) deixa de haver a preocupação de possivelmente se estar a apagar código que mais tarde vamos perder tempo a reproduzir e que afinal era mesmo importante. Com um bom VCS podemos voltar sempre a qualquer snapshot do código, inclusivamente a esse momento anterior a termos apagado código. Esta característica também é útil no backup de dados, porque se os programadores guardarem regularmente as suas alterações no repositório central (no SVN, isto designa-se de commit) que tem todo o código, então é fácil ter um processo automático que arquive todo este código do repositório, e assim resolve-se o problema do código não estar unicamente no computador de um programador (mesmo à espera que aconteça aquela falha inevitável no disco...).

2. **Produtividade** – Ao libertar os programadores da tarefa árdua de ter que integrar manualmente todas as alterações que fizeram (e tipicamente em muitos ficheiros ao mesmo tempo), um VCS pode aumentar bastante a productividade na criação de *software*. De facto, com um bom VCS os programadores conseguem testar as alterações que fizeram e que outros programadores fizeram, resolver possíveis conflitos de código (que correspondem a mudanças nos mesmos ficheiros e nas mesmas linhas desses ficheiros) mesmo antes de incluir esse código no repositório central, e de uma forma automática. E, com mais tempo para programar, nasce melhor *software* (se o programador não se distraír das suas programações).

3. **Gestão de Projectos** – Um bom VCS consegue reunir rapidamente um conjunto de informações muito úteis para um gestor de projectos: quem alterou o quê, quando, porquê, quantas modificações fez no mesmo projecto (ou módulo), e se as alterações feitas vão de encontro à finalidade do projecto (por exemplo, se existirem várias modificações que não resolvem os bugs encontrados). Estas informações são importantes para os gestores de projectos e para os programadores que não são apenas programadores mas também líderes de pequenas equipas de *software*. Apesar de um VCS estar principalmente preocupado com a gestão das versões, as informações que fornece podem ser facilmente usadas por *software* mais específico da área de gestão de projectos.

4. **Suporte a projectos de Engenharia de *software*** – Tipicamente, bons projectos de *software* são desenvolvidos com um processo de engenharia de *software*. Por engenharia de *software*, quero dizer a aplicação de políticas de desenvolvimento com o intuito de assegurar que o producto final do processo vai de encontro aos objectivos, dentro dos prazos previstos e com os melhores padrões possíveis de qualidade do *software*. Um bom processo de engenharia de *software* envolve um conjunto de passos, tais como o desenho bem detalhado do projecto na sua globalidade, a revisão dos componentes do *software* por parte dos analistas e engenheiros, a anotação de todos os bugs, e testes de qualidade ao *software*. Nenhum destes passos é explicitamente suportado por qualquer VCS actualmente, mas as características de muitos VCS (como por exemplo, os *hook scripts* e *logs* do SVN) são excelentes ajudas ao processo de engenharia de *software*. Vamos ver mais adiante neste anexo o que o SVN nos fornece neste campo.

5. **Ramificação do desenvolvimento** – À medida que os projectos vão aumentando, há necessidade de ramificar o projecto. Isto é, vai surgir naturalmente a necessidade de um programador (ou conjunto de programadores) desenvolver features que não podem de maneira nenhuma minar o trabalho que já existe guardado num repositório de código central. Por exemplo, basta pensar num novo módulo que vai demorar tempo a ser desenvolvido e que vai ter repercussões nos restantes módulos. Neste caso, o ideal é o programador trabalhar num ramo privado, em que este ramo não é mais do que uma cópia de todo o código que já existia e no qual vai poder alterar à vontade, sem o problema de estar a interferir com o código de outros programadores. No final, quando todas as alterações estiverem feitas, o programador funde todo o seu código com o que existe no repositório central (podendo ter que resolver conflitos), e o ramo pode deixar de existir. Outro caso em que os ramos são importantes é no suporte a versões antigas do *software*. Cada uma das versões antigas poderá ser um ramo do código a dada altura (por exemplo, na altura em que sai uma nova versão), e a equipa de programadores pode testar a resolução dos bugs

nesses ramos de código antigo e posteriormente entregar ao cliente uma versão corrigida desse *software* para aquela versão anterior. Nem todos os VCS possuem esta funcionalidade nem tão pouco fornecem uma alternativa. No entanto, o VCS analisado neste artigo fornece esta a outras funcionalidades importantes para o controlo do *software*.

6. **Anotação dos logs** – Para além de saber quem alterou o quê, é também importante saber o porquê. A cada alteração deverá ficar sempre associada uma mensagem que traduza a alteração que se fez e que pode posteriormente ser usada para construir um `CHANGELOG` ou ainda ser injectada num sistema de tracking, tal como o Trac [Sof09c] de que falaremos mais abaixo.

7. **Distribuição de trabalho** – Actualmente, com a era da globalização e do alcance fácil de uma ligação à Internet, o trabalho à distância é uma realidade presente em cada vez mais empresas, seja o programador que está a aceder ao código do outro lado da cidade ou seja uma empresa sub-contratada do outro lado do planeta; um VCS deverá conseguir equilibrar esta distribuição de trabalho ao gerir todas estas alterações ao código pelos diferentes programadores e também unificar este mesmo código automaticamente. Combinando estas características com uma comunicação (que é fundamental) através de e-mail ou um cliente de IM (como o GTalk , MSN Messenger , Skype ou ICQ), é praticamente tão fácil de trabalhar remotamente como localmente ao lado dos restantes elementos da equipa.

8. **Desenvolvimento rápido** – As mais recentes metodologias de desenvolvimento de *software* apontam para uma forma de programar mais rápida, e flexível usando metodologias de *Extreme Programming* (XP) e de *Desenvolvimento Ágil* de *software*. Estas metodologias de desenvolver *software*, esperam por iterações mais rápidas e sucessivas relativamente a metodologias mais ”planeadas”. Esta forma de programar é facilitada por um VCS que consiga manter e indicar todas as alterações por que passou o *software* nas sucessivas iterações. Para além disso, um repositório central de *software* pode ajudar na automatização dos *builds* frequentes que são precisos por um processo de desenvolvimento ágil.

Por todas estas razões, tornam-se evidentes as vantagens na adopção de um VCS no controlo de projectos informáticos[10].

2.4 Lista de Sistemas de Controlo de Versões

Actualmente, os SCM's encontram-se agrupados numa das categorias possíveis mencionadas acima e indicadas na Figura 2.3:

[10] Os projectos informáticos que se mencionam aqui não têm necessariamente de ser projectos de *software*.

Figura 2.3: Categorias de SCM's. FOSS (*Free and Open Source Software*) significa *Software* livre e de *Código-Aberto*.

Para além de um VCS poder ser do tipo centralizado ou distribuído, pode ainda ser de utilização livre ou através de uma licença paga (o dito *software* comercial). Dentro destas categorias, destacam-se actualmente os seguintes:

- **Sistemas Centralizados e Comerciais**:
  - Perforce;
  - IBM Rational ClearCase;
  - Borland StarTeam;
  - Microsoft Visual Studio Team System;
  - Microsoft Visual SourceSafe;
- **Sistemas Centralizados e FOSS**:
  - CVS (*Concurrent Versions System*);
  - Subversion (também designado por SVN);
- **Sistemas Distribuídos e Comerciais**:
  - BitKeeper;
  - Code Co-op;
- **Sistemas Distribuídos e FOSS**:
  - Git;
  - Bazaar;
  - Mercurial;
  - Monotone;
  - SVK;
  - Darcs;
  - GNU Arch.

2.5 Comparação entre Sistemas de Controlo de Versões

No anexo C listam-se as principais características entre os SCM's listados em cima, separados pelo tipo: centralizado e distribuído.

2.6 Integração com outras ferramentas

Alguns SCM's mais avançados oferecem outras facilidades, tornando possível uma integração mais profunda com outras ferramentas e nos próprios processos de engenharia de software. Há vários *plugins* disponíveis para alguns *IDE's* como o IntelliJ IDEA [Jet09], Eclipse [Com09] e Visual Studio [Cor09d]. Os *IDE's* Netbeans [Mic09] e Xcode [Inc09] (Mac OS X) já trazem suporte embutido para controlo de revisões. Existem ainda ferramentas que se integram completamente com os ambientes de trabalho como o TortoiseSVN [Col09e] (para Windows) ou o NautilusSVN [Goo09b] (para Linux).

2.7 Conclusões

Este capítulo descreveu o estado-da-arte dos SCM's e as ferramentas disponíveis que se integram com os principais SCM's.
No capítulo seguinte é descrito o processo anterior de desenvolvimento de *software* na equipa de Desenvolvimento da divisão de I&D, para daqui se retirar o problema que a dissertação se propõe a resolver. No capítulo 4 são apresentadas as metodologias propostas para solucionar este problema.

If I had 8 hours to chop down a tree, I would spend 6 hours sharpening an axe.

Anónimo

3

Desenvolvimento na EFACEC: um Caso de Estudo

Este capítulo descreve o modelo de controlo de versões de *software* anteriormente adaptado no departamento de I&D da EFACEC Engenharia, na divisão de Gestão de Redes, constituindo um caso de estudo para daqui derivarmos as metodologias propostas para resolução dos problemas deste modelo, no Capítulo 4.

3.1 Visão Geral

O controlo de versões no Departamento de I&D da EFACEC era muito inseguro e sujeito a erros graves por parte da equipa de desenvolvimento, mesmo em acções tão casuais como a escrita de um ficheiro, como veremos em seguida na secção 3.2. Através do estudo realizado na dissertação, foi possível assegurar um controlo de revisões seguro, escalável, fiável e, como veremos no capítulo 4, disponível para automatizar tarefas de *build* e *deployment* de artefactos para os diversos clientes, aumentando desta forma a produtividade na equipa de desenvolvimento.
Actualmente existem na divisão de I&D os seguintes produtos: SCATEX, GENESys EDP e SXP2, SX-SEE, SAH e HIM, BUS e FE (ver caixa *Gestores de Produtos* na Figura 3.1). Cada um destes produtos são desenvolvidos nas linguagens de programação `C` (com recurso ao *toolkit* `X/Motif`), `C++`, e/ou `Java`.

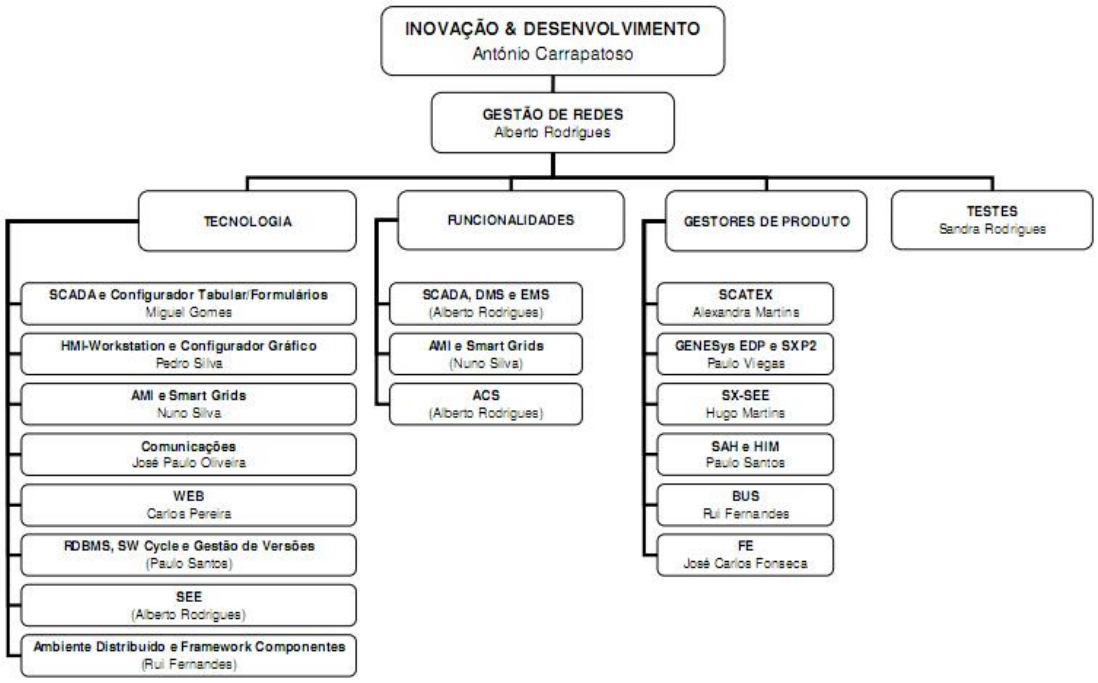

Figura 3.1: Organograma da Unidade de Negócio Automação, Departamento de I&D, Divisão de Gestão de Redes.

Destes 8 produtos, 4 não tinham qualquer tipo de versionamento (SCATEX, FE, SXP2 e SX-SEE) e os restantes estavam a ser versionados com o Microsoft Source Safe [Cor09b]. A dissertação incidiu exclusivamente no produto SCATEX, pelo que o foco principal da dissertação recairá sobre o versionamento deste produto.

3.2 Desenvolvimento no SCATEX

O SCATEX é actualmente um produto SCADA com cerca de 12100 ficheiros, 1070 pastas e cerca de 320 MB de código-fonte não compilado. Por estes números pode-se perceber o perigo que existia em não possuir nenhum sistema de controlo de versões sobre o código deste sistema SCADA.

O ciclo de desenvolvimento no SCATEX começava com o registo de um *bug* na ferramenta de gestao de *bugs*, *Project Link* (ou PL). A partir desta ferramenta obtinha-se um identificador único para o *bug* em questão (por exemplo, PL1234). Dava-se então início ao desenvolvimento propriamente dito, a partir de acessos remotos a uma única máquina ”mãe”que continha todo o código-fonte, e a partir desta máquina cada *developer* copiava o código-fonte para uma máquina com um sistema operativo específico (máquinas de testes), para poder compilar o código-fonte e poder testá-lo nesse sistema operativo específico (Figura 3.2).

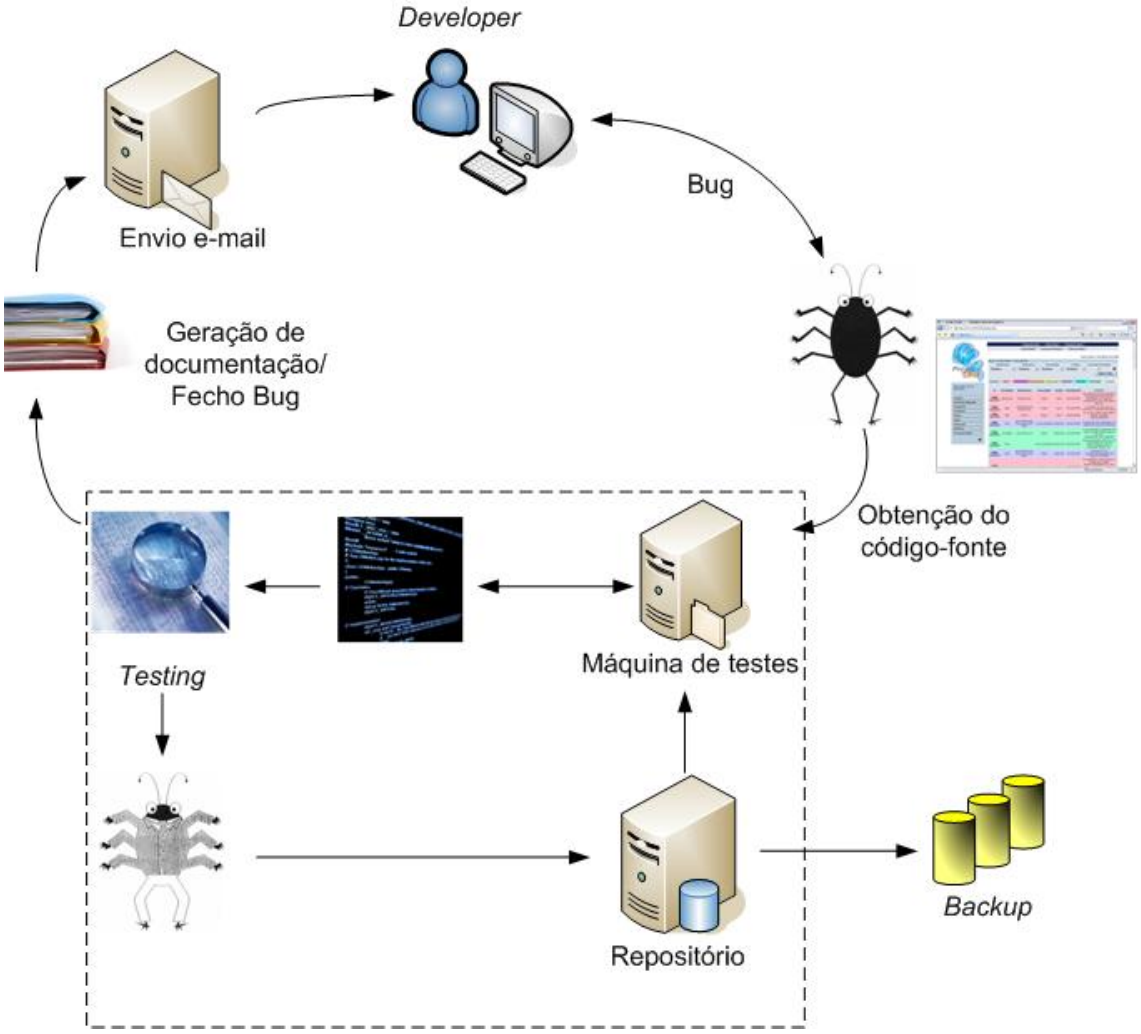

Figura 3.2: *Workflow* de desenvolvimento no SCATEX.

O código-fonte era exclusivamente alterado no máquina ”mãe”por todos os *developers* concorrentemente. Não existia nenhuma forma segura de controlo dos acessos para editar e alterar o código-fonte na máquina ”mãe”. Quando os testes terminavam, o código-fonte da máquina ”mãe”continha desta forma a correcção ou a funcionalidade pedida pelo cliente. Por fim, usava-se a ferramenta de gestão de *bugs* para fechar o *bug* em questão (o que despoletava o envio de um e-mail à comunidade de *developers*) e o ciclo de trabalho fechava de novo junto dos *developers*.

Em termos de casos de uso, o cenário para o desenvolvimento e testes de produto era simples: cada *developer* desenvolvia de acordo com os *bugs* ou *features* a incluir, mediante o consentimento do gestor de projecto para tal. No fim da correcção feita pelo *developer*, o código-fonte seria compilado (sem erros) e feitos os primeiros testes superficiais na alteração acabada de realizar. Por fim, estes artefactos seriam disponibilizados à equipa de testes para os testar e validar numa máquina da rede preparada para o efeito (Figura 3.3).

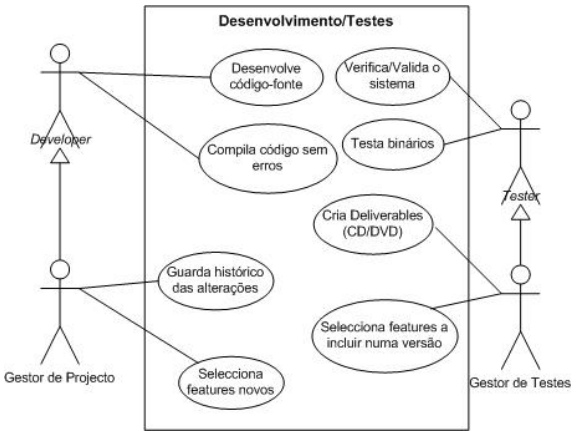

Figura 3.3: Casos de uso no desenvolvimento do produto SCATEX.

Se tudo estiver conforme o esperado, o gestor da equipa de testes cria um *deliverable* (CD ou DVD com os binários e ficheiros de configuração) para ser entregue ao cliente com determinada versão do produto.

3.3 Documentação

No que toca ao cenário para a documentação, o caso de uso é igualmente simples: cada *developer*, *tester* ou gestor de projecto começa por criar um PL (*Project Link*), que corresponde à criação de um novo ID para associar a um *bug* ou uma nova funcionalidade pretendida, como vimos anteriormente.
De seguida, é criada a documentação associada ao PL (no designado SGD[1]) e assim que estiver terminada, é colocado o PL em revisão. Esta acção provoca o envio automático de um e-mail às equipas de desenvolvimento e de teste a indicar que um novo documento está em revisão, para que seja revisto e debatido pela equipa (ou parte desta). Por fim, se tudo estiver conforme o pretendido, o PL é resolvido, o que provoca igualmente o envio automático de um e-mail a indicar a passagem para o estado de resolvido do PL (Figura 3.4).

3.4 Imputação de horas

Para controlo das horas dispendidas por cada actor envolvido no desenvolvimento e testes do produto, recorre-se a uma *timesheet* (folha de cálculo em Microsoft Excel com um determinado *template* para a *timesheet*), para guardar todas as horas gastas por cada tarefa de trabalho. No final de cada semana, as folhas de cálculo deverão estar prontas, para que

[1]Sistema de Gestão Documental. É um servidor contendo toda a documentação associada a correcções, novas implementações, produtos, estudos e obras por cada departamento existente.

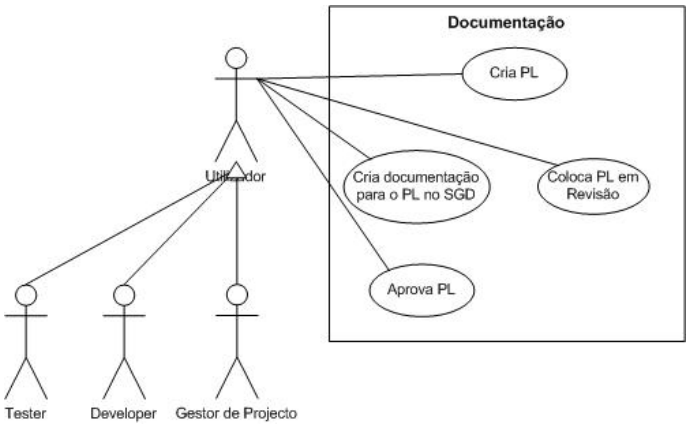

Figura 3.4: Casos de uso na geração de documentação do produto SCATEX.

se possa avaliar efectivamente o trabalho de cada actor e o que ainda resta fazer. Esta tarefa é essencial para um correcto planeamento dos projectos de desenvolvimento.
No final de cada semana, é ainda feita a imputação de horas no sistema BAAN ERP para controlo das horas gastas por cada membro do departamento de I&D. Corresponde a uma cópia do que é imputado nas *timesheets*, mas é usado para uma gestão mais completa e fiável dos produtos desenvolvidos.

3.5 Salvaguarda do código-fonte

Em termos de restauro e salvaguarda de todo o código-fonte e artefactos criados a partir do mesmo, eram feitas cópias para *tape drives* todas as semanas, em dias específicos através do processo `cron`, configurado na máquina ”mãe”para realizar a construção dos arquivos `.tar.gz` do código-fonte. Quando surgiam problemas de disco, era necessário proceder a uma extracção do arquivo `.tar.gz` da *tape drive*, o que iria eliminar qualquer código-fonte que tivesse sido alterado antes do problema no disco da máquina ”mãe”.

3.6 Conclusões

Este procedimento de desenvolvimento era pouco confiável e continha diversas falhas: não era intuitivo para o *developer*, rápido (as cópias bi-direccionais de código-fonte não são rápidas, especialmente com problemas de rede casuais), e fiável porque não existia qualquer tipo de controlo entre o código-fonte que saía numa direcção e que entrava noutra direcção.
Como seria de esperar, não era difícil de surgirem problemas com as escritas concorrentes de ficheiros, a acumulação de ficheiros temporários na máquina ”mãe”para salvaguarda do código já existente, antes de uma edição por parte do *developer*, era inevitável e os

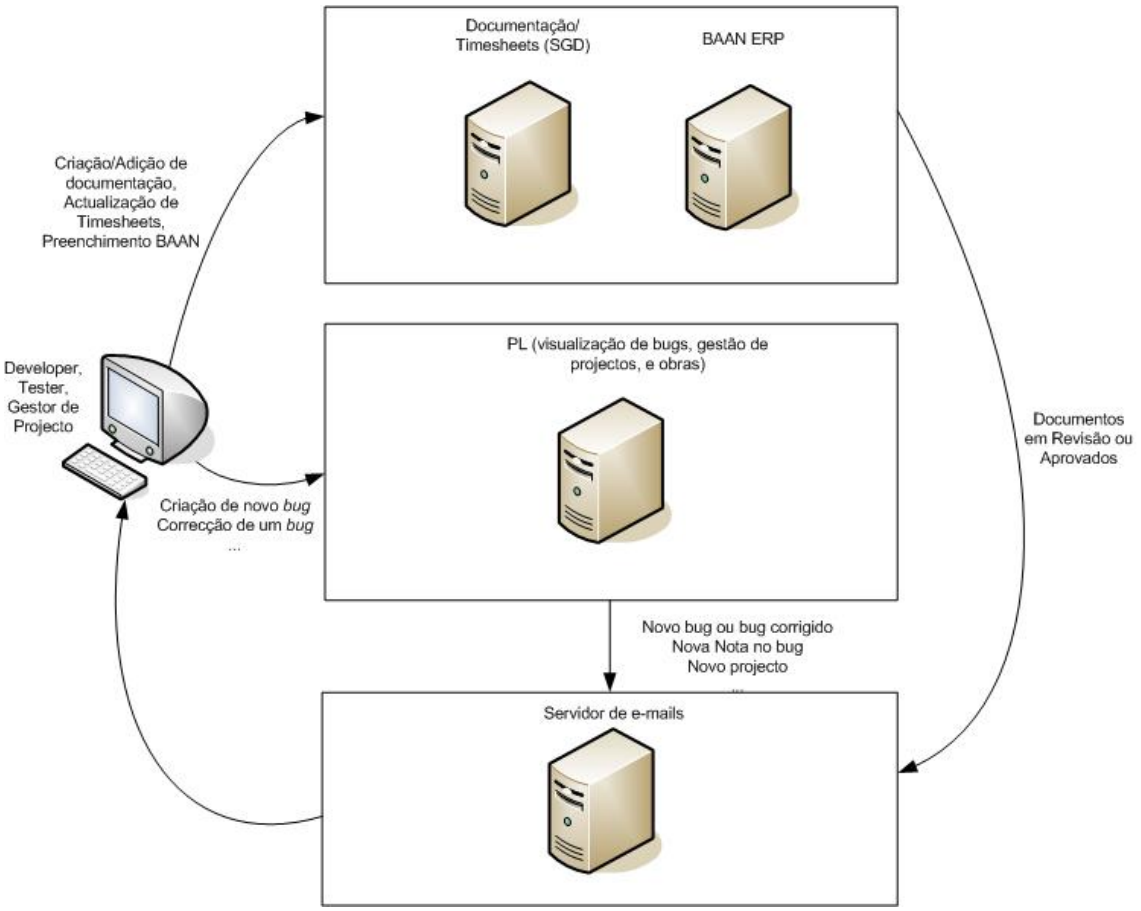

Figura 3.5: Visão de alto nível sobre a documentação, *timesheets*, e gestão de *bugs* ou funcionalidades.

problemas com os discos poderiam ser críticos, uma vez que o único ponto na rede que continha o codigo-fonte mais actual era a máquina ”mãe”.

Relativamente à documentação, a sua geração não corresponde a um processo automático: é necessário copiar um *template* para a criação de uma nova documentação de um *bug* ou funcionalidade, o que é uma perda desnecessária de tempo. Para além disso, não é portável, uma vez que o *template* baseia-se na ferramenta Microsoft Word, que só funciona em ambientes Windows. O envio automático de um e-mail (ver Figura 3.5) baseia-se igualmente num script que só funciona em ambientes Windows. O utilizador que pretender usar outro sistema operativo não pode gerar de forma simples a documentação e o envio de e-mails para os restantes membros da equipa[2].

Em termos gerais, a geração de documentação e a gestão de *bugs* e correcções na divisão de Gestão de Redes, para o produto SCATEX, pode ser visto na Figura 3.5, que contempla o que já foi descrito nas secções acima.

[2]Uma possibilidade para circunscrever este problema é a utilização de uma máquina virtual Windows, mas esta alternativa consome bastantes recursos à máquina do utilizador, e obriga a possuir uma licença válida Microsoft para uso do sistema operativo Windows

Ter um sistema complexo como um sistema SCADA sem qualquer VCS que auxilie na gestão das alterações ao código corresponde a uma insegurança grave e sujeito a múltiplos pontos de falha. Estes problemas são facilmente resolvidos com a adopção de um VCS. Portanto, o primeiro grande ponto de partida seria o de escolher um VCS e adaptá-lo ao código-fonte e à forma de trabalho nas equipas de desenvolvimento e de testes. Como segundo objectivo, existiu sempre a vontade de não usar o VCS apenas como um VCS: estudar a possibilidade de o extender, disponibilizar funcionalidades integradas com o VCS às equipas de desenvolvimento e de testes, e de aferir se a produtividade era superior desta forma.
No capítulo seguinte (capítulo 4), é feita uma análise do trabalho correspondente primeiramente à adopção do VCS usado para controlo das versões do produto SCATEX – SVN – e em seguida, das metodologias propostas para sistematizar e melhorar o ciclo de desenvolvimento de *software* recorrendo unicamente a ferramentas *open source*.

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

D. Knuth

4 Metodologias Propostas

Neste capítulo é feita a apresentação da solução proposta para a resolução dos problemas relatados no Capítulo 3, sob a forma de duas possíveis metodologias, que serão detalhadas neste Capítulo. São também apresentadas as razões para a escolha do SVN como VCS do produto SCATEX.

4.1 Escolha do SVN

A escolha do SVN como VCS foi relativamente simples e unânime: é um sistema livre, com binários para os sistemas operativos mais utilizados (Windows, Linux, Unix, Mac OS), muito popular por todo o Mundo, com mais *plugins* e ferramentas do que outro VCS, e é um VCS centralizado, o que reúne mais condições de segurança, uma melhor gestão dos repositórios locais e de aceitação dos próprios *developers* do que no modelo distribuído. Para além destes motivos, as vantagens na adopção do SVN são em grande número e fornecem qualidade para o utilizador, o que de sobremaneira facilitou a escolha deste VCS.

4.2 *Layout* do repositório de SVN

Com a escolha do SVN, foi necessário pensar numa estratégia para o *layout* do repositório que iria receber toda a estrutura de directórios e de ficheiros do produto SCATEX. Uma vez que a equipa de testes também iria usar o produto SCATEX, foi necessário também ter em conta com este facto no desenho do *layout* para o repositório.
Com estes requisitos em mente, o repositório foi configurado com o seguinte *layout* (Figura 4.1):

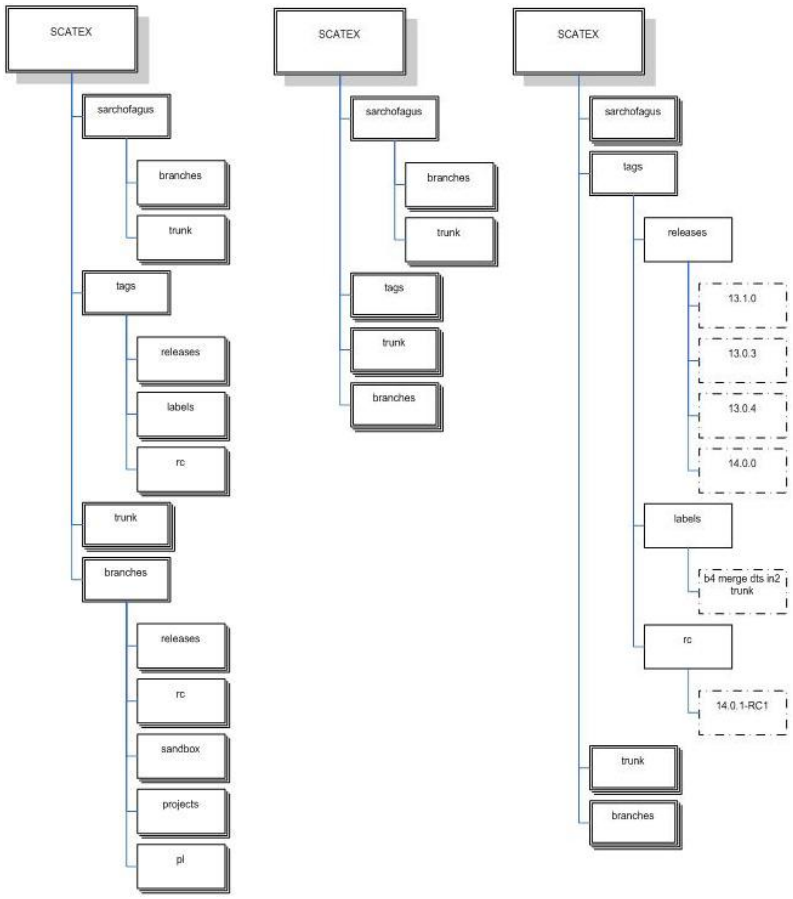

Figura 4.1: *Layout* do repositório de SVN para o produto SCATEX. Fonte: [MR09].

Em termos de *layout* lógico, esta estrutura tem a seguinte organização:

- ***sarchophagus*** - esta área pretende conter o conjunto de artefactos que não estão num estado activo, ou porque um ramo `pl/project` já terminou o desenvolvimento pretendido e já foi integrado completamente, ou porque um conjunto específico de artefactos do `trunk` ficou obsoleto; Esta área subdivide-se nas seguintes (Figura 4.1):
  - ***branches*** - contém o conjunto de artefactos inactivos originários da área *branches*;

  - ***trunk*** - contém o conjunto de artefactos inactivos provenientes da área *trunk*;

- ***tags*** - esta área contém todas as *tags* feitas para marcar um novo *checkpoint* no produto SCATEX; Esta área subdivide-se nas seguintes (Figura 4.1):

  - ***releases*** - é a área de *tags* usadas para marcar a versão de uma *release* final;
  - ***rc*** - é a área de *tags* usadas para marcar uma versão *release candidate* para ser entregue à equipa de testes;
  - ***labels*** - corresponde a uma área *tags* para marcar um *checkpoint* no ciclo de vida do produto diferente das *tags* `releases` ou `rc`;

- ***trunk*** - contém o conjunto de artefactos mais actual do repositório que define o produto (excluindo os binários de *runtime*), com o seu histórico de alterações; Não tem subdivisões como nas restantes áreas, apenas contém todo o código-fonte (Figura 4.2);

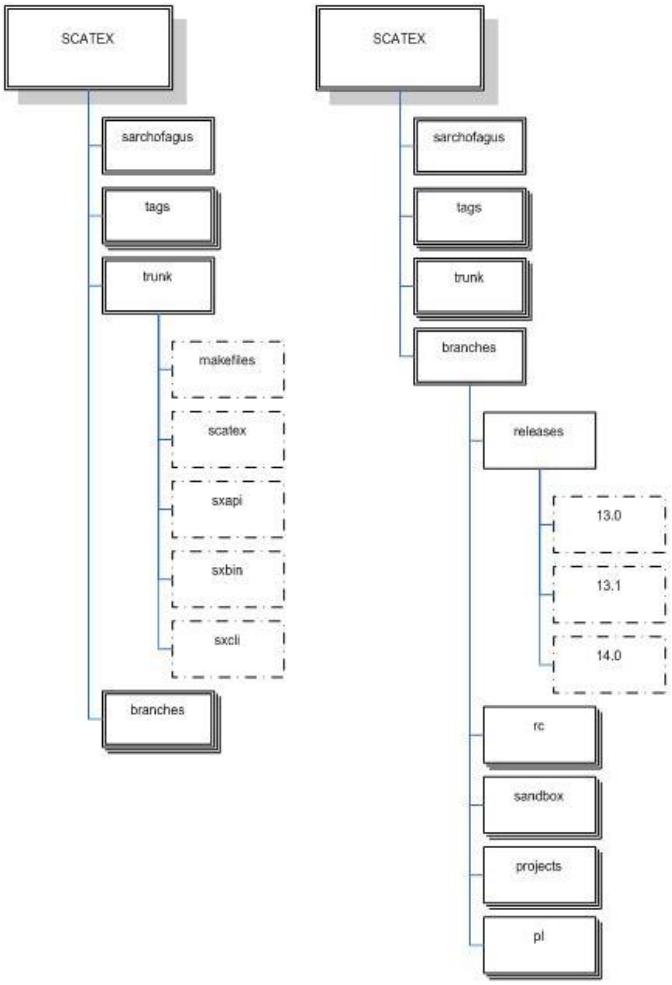

Figura 4.2: *Layout* do trunk e branches no repositório de SVN. Fonte: [MR09].

- ***branches*** - contém os ramos de desenvolvimento privados e de experiências pessoais (*sandbox*); Esta área subdivide-se nas seguintes (Figura 4.2):
  - ***releases*** - é um ramo de manutenção para cada *release major/minor* feita;
  - ***rc*** - é um ramo *release candidate* que contém cada versão *release candidate* que está em fase de preparação ou validação para se tornar uma *release* final; Uma vez disponibilizada como uma nova versão, o ramo *rc* correspondente é unificado com um *merge* ou directamente movido para o ramo *releases*;
  - ***sandbox*** - é a área que contém ramos pessoais não oficiais de desenvolvimento; Estes ramos deverão ser removidos assim que não sejam utilizados por nenhum elemento da equipa;
  - ***projects*** - contém os projectos de desenvolvimento de longo termo que podem entrar em conflito com o desenvolvimento regular de um produto; Uma vez completos ou unificados no *trunk*, e numa *release*, o projecto deverá ser movido para a área *sarchophagus*;
  - ***pl*** - contém os projectos de curto termo criados para corrigir um problema (PL) ou adicionar uma nova funcionalidade; Uma vez completos ou unificados no *trunk*, e numa *release*, o ramo *pl* deve ser movido para a área *sarchophagus*.

4.3 Mudanças no Workflow

O *workflow* foi igualmente revisto para contemplar um modelo mais simples e ágil de desenvolvimento (ver Figura 4.3).
O desenvolvimento passa agora por usar o repositório de SVN para obter todo o código-fonte (operação com um custo muito menos acentuado que uma cópia por rede do projecto SCADA), e permite igualmente que o *developer* possa efectuar testes na sua própria máquina de desenvolvimento. Para além disso, o tempo de construção de uma área local de trabalho e de compilação diminui consideravelmente com a adopção deste novo *workflow*. Sucintamente, o desenvolvimento passa pela obtenção do código-fonte, criação de um ramo de desenvolvimento (para proceder à correcção e testes num ramo separado do desenvolvimento actual), e por fim, integração dessa correcção no ramo de desenvolvimento e noutros ramos importantes do repositório.
Finda a fase de correcção do *bug* em questão, o *workflow* passa pelas fases vistas anteriormente: fecho do *bug*, criação da documentação associada à correcção em causa e posterior envio automático de e-mail para conhecimento de toda a equipa de desenvolvimento/testes.

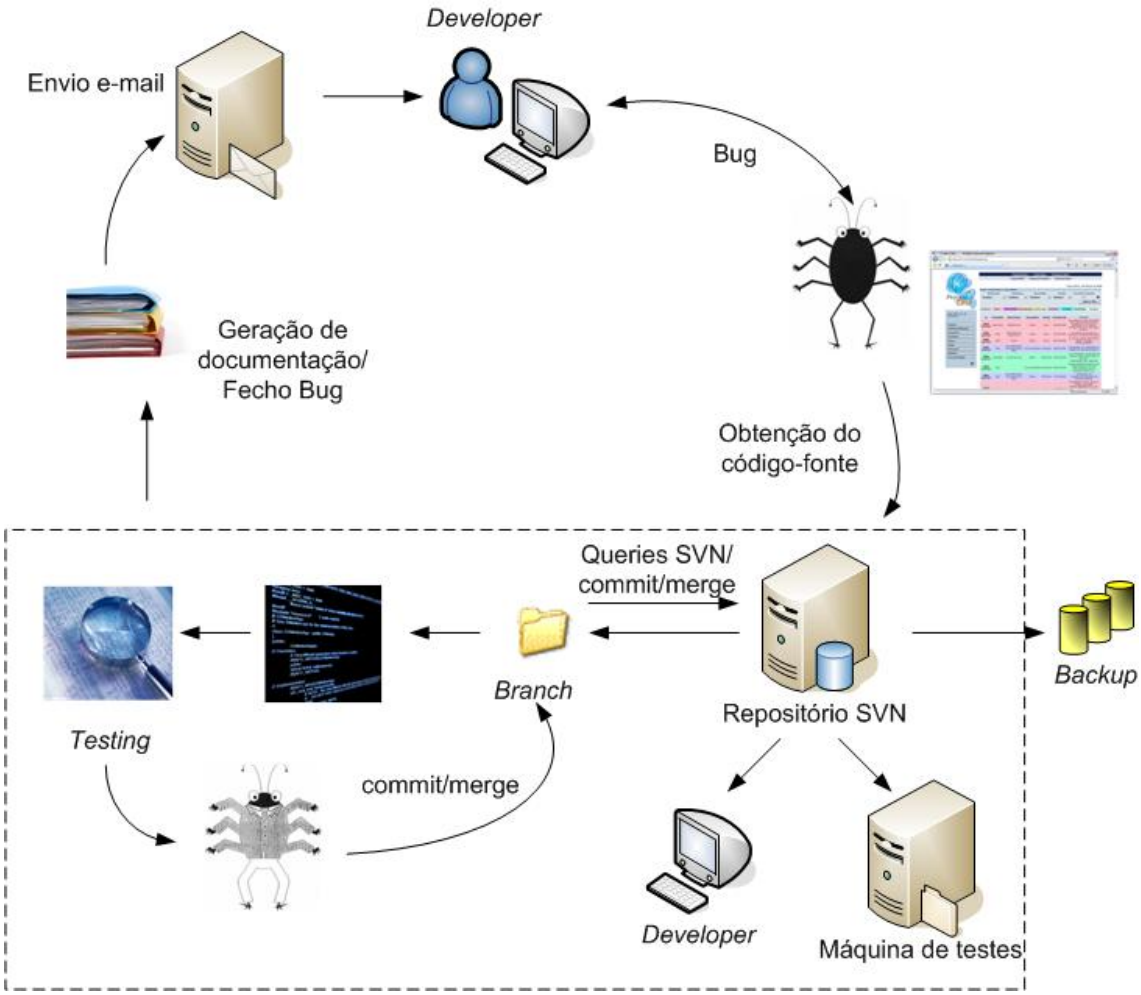

Figura 4.3: *Workflow* melhorado no ciclo de desenvolvimento de *software* do SCATEX.

4.4 Ferramentas *open source*

Do leque actual de ferramentas *open source*, há a destacar 5 que se integram completamente com o SVN e permitem aumentar significativamente a produtividade dentro de uma equipa de desenvolvimento. Vejamos as ferramentas seleccionadas e o porquê de serem recomendadas:

- **Eclipse IDE** [Com09]: é um IDE livre, fácil de extender, com suporte a múltiplas linguagens de programação (C, C++, Java, Python, PHP, Perl, XML, etc.), rápido, com uma quantidade impressionante de *plugins* implementados por equipas especializadas por todo o Mundo e para além disso, tem suporte aos SCM's mais usados hoje em dia: CVS, ClearCase, Perforce, e é claro, SVN. É um IDE de eleição e uma escolha clara para a equipa de desenvolvimento do produto SCATEX;
- **Mylyn** [Ker09]: é um *plugin* para o Eclipse que auxilia o *developer* na sua tarefa de corrigir determinado *bug*, aumentando a sua produtividade ao não fazer perder

tempo com situações acessórias. A saber:

- **Foca o developer na tarefa concreta que está a realizar**: a cada nova tarefa é definido o conjunto de actividades para essa tarefa e o tempo estimado de conclusão. Cada tarefa é categorizada e à medida que surgem novos bugs para resolver, o workbench tem várias categorias diferentes de tarefas que ajudam o developer a tomar nota do progresso das suas tarefas (se muito adiantadas ou atrasadas, por exemplo) e do que ainda falta fazer no prazo definido pelo developer. Esta Task List pode ainda ser filtrada para mostrar as tarefas já concluídas ou esconder tarefas abaixo de uma certa importância (*Very High*, *High*, *Normal*, *Low* ou *Very Low*). Cada tarefa tem associado um conjunto de informações: agendamento da tarefa, ou seja, o tempo que se espera que uma tarefa vá demorar, quando é que o developer estima que vá começar a trabalhar nessa tarefa (2.ª feira, 3.ª feira, ...) e quanto da tarefa já foi cumprida. Estas informações formam o contexto da tarefa.
- **Indica apenas conteúdo que interessa ao developer**: o Mylyn cria um contexto no workbench Eclipse que toma nota dos ficheiros, tipos, métodos, etc., que o developer abre e/ou altera, e constrói este contexto a partir dos últimos artefactos vistos pelo developer. Tudo o que o developer usa é mostrado e tudo o resto é ocultado. À medida que se vai usando o worbench os artefactos mais vistos e alterados vão ganhando realce (por exemplo, as classes Java ficam com nome a bold e os items normais com o tempo ficam cinzentos e após mais tempo ainda desaparecem da vista outline ou do package explorer se for uma classe). Este comportamento do Mylyn pode ser disabled no botão "*Focus on Active Task*". Outra vantagem é que se pode explorar artefactos não relacionados com a tarefa em curso sem encher o workbench com esses artefactos que não têm nenhuma relação com uma tarefa específica (suspendendo a tarefa actual). Esta característica é nativa do Mylyn para Java, mas pode ser usada noutras linguagens com a API do Mylyn;
- **Recorda o que o developer estava a fazer quando muda entre tarefas**: só pode haver uma tarefa activa a cada instante, mas se se mudar de tarefa o Mylyn mostra o estado a que estava uma tarefa quando se muda entre tarefas. Ao ter um contexto para cada tarefa, o Mylyn permite ao developer mudar rapidamente e facilmente entre tarefas e pô-lo a par do que estava a ser feito em dada tarefa, poupando assim tempo precioso;
- **Suporte a integração com Bugzilla, JIRA, Trac e XPlanner**: Ao ligar o Mylyn com o Trac pode-se expôr à equipa de desenvolvimento o que cada *developer* está a fazer a dado momento com todas as vantagens que o Trac como sistema de *bug tracking* fornece. Para além disto, é possível manipular

as tarefas (*bugs*) em *offline* e posteriormente essas tarefas (*bugs*) são sincronizadas com o Trac na ligação seguinte. Estas tarefas se forem alteradas no Trac são marcadas pelo Mylyn, o que indica visualmente ao *developer* que houve mudanças no estado do bug no Trac, tudo dentro do workbench do Eclipse. É ainda possível anexar o contexto de uma tarefa a uma tarefa no repositório, que permite a outros developers verem o conjunto de artefactos com que se estava a trabalhar no momento em que se fez o anexar do contexto. Outros *developers* que trabalhem na mesma tarefa, podem obter este contexto usando a opção de "*Retrieve Context*"e ter o mesmo contexto no workbench. Esta característica para trabalho colaborativo é muito interessante e aumenta a colabaração entre *developers*;

  - **Possibilidade de extender o Mylyn**: através da API pública do Mylyn pode-se construir conectores adicionais para outras fontes de tarefas, incluindo *web services*, bases de dados, entre outros;
  - **Conectores adicionais**: estão disponíveis extensões para o Mylyn, nomeadamente integração com Microsoft Outloook, Xplanner, e Google calendar.

- **ECF**[Fou09b] - é uma *framework* de comunicação para o IDE Eclipse, que permite que os utilizadores do Eclipse possam comunicar com outros utilizadores dentro do próprio IDE, usando para isso o protocolo P2P [Wik09m]; Permite o uso de ferramentas de comunicação como o MSN Messenger [Cor09a] ou o Skype [Lim09] no workbench do Eclipse;

- **SVNKit** [Sof09e] - É uma biblioteca pura Java que implementa o Subversion. Com esta biblioteca pode-se criar e customizar aplicações Java para usar o Subversion de forma a maximizar a sua utilização no seio de uma equipa de desenvolvimento (ver secção 4.5);

- **Trac** [Sof09c] - O Trac [Sof09c] é um sistema de gestão de projectos e de *bug-tracking* que integra numa wiki o conteúdo de repositórios SVN para uma rápida visualização do que se passa nos mesmos. Para além de possibilitar aceder a todo o código de um repositório SVN (com *highlight* de código), permite introduzir na wiki os objectivos e os prazos para o projecto, os *bugs* encontrados, aferir o que cada utilizador tem alterado no projecto, entre outras funcionalidades muito interessantes para uma visualização intuitiva e rápida dos projectos. Este sistema, que para além de ser *open source*, ajuda no controlo e na análise das alterações de um projecto: possibilita ver que modificações foram feitas a dada revisão, ver diferenças entre diferentes projectos que estão num mesmo repositório, e descarregar o conteúdo do projecto num arquivo `.zip`. Para além de que é uma wiki com tudo o que de positivo existe no trabalho colaborativo numa wiki.

- **Bitten** [Sof09a] - é uma *framework* escrita em Python [Fou09c] para coleccionar várias métricas de software através de uma contínua integração no Trac. Permite automatizar a construção de *builds* de sistema e a criação de arquivos com os binários de *runtime* para se incluir num *deliverable* (CD/DVD) a um cliente.

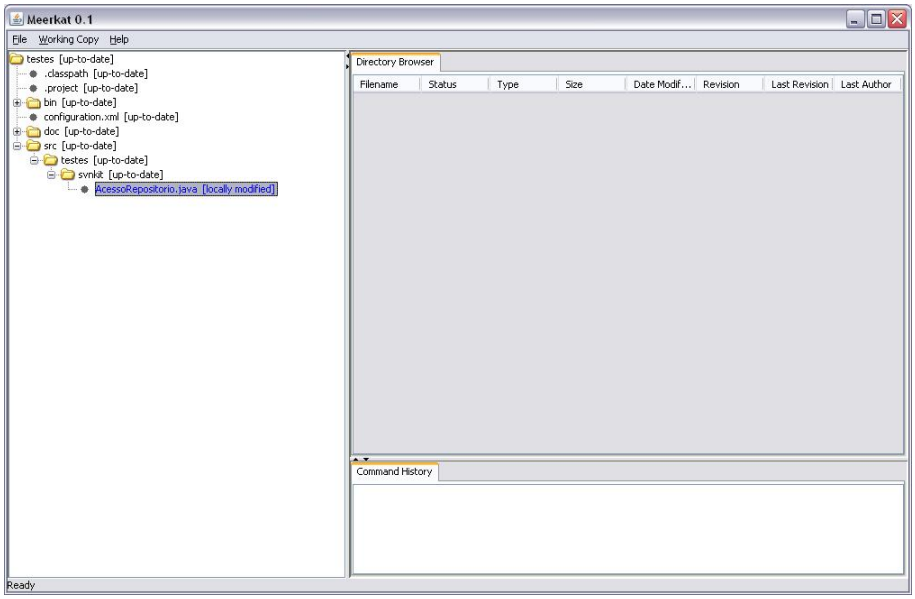

Figura 4.4: Janela principal da aplicação *Meerkat*.

4.5 Metodologias Propostas

Face ao estudo realizado dos VCS actualmente mais utilizados, e das ferramentas que mais tiram partido dos mesmos, foi possível delinear duas metodologias para aumentar a produtividade no modelo de desenvolvimento do departamento de I&D da EFACEC Engenharia:

- **Uso de ferramentas *open source*** - instalação das ferramentas indicadas acima, e posterior customização para uma melhor adaptação no ciclo de desenvolvimento da equipa de desenvolvimento e de testes, sem provocar dificuldades aos *developers* ou alterações significativas na forma habitual de trabalho;

- **Programação de uma aplicação Java** - outra metodologia possivel é a programação uma aplicação Java com a biblioteca SVNKit para gerir todas as alterações das cópias locais de trabalho de cada *developer*. Com esta aplicação seria mais simples a integração com o actual modelo de desenvolvimento e de geração de documentação e *timesheets*, uma vez que se poderia programar a aplicação para automatizar certos processos específicos do departamento de I&D da EFACEC Engenharia, como vimos na secção 3.2 (ver Figura 4.4).

Estas duas metodologias foram implementadas com sucesso, sendo que a primeira (uso de ferramentas *open source*) revelou-se a melhor opção, por diversos motivos:

- Quantidade de ferramentas disponíveis hoje em dia para responder às necessidades pretendidas pela equipa de desenvolvimento;
- Facilidade em integrar essas ferramentas com o actual modelo de desenvolvimento;
- Robustez e escalabilidade das ferramentas (o produto SCATEX poderia duplicar de tamanho que as ferramentas teriam igualmente um comportamento fiável, embora perdessem possivelmente alguma velocidade no processamento);
- O *plugin* Mylyn e a ferramenta *Trac* revelaram-se excepcionalmente vantajosas para a equipa de desenvolvimento, o que permitiu aumentar de sobremaneira a rapidez de correcção de *bugs*, visualização de correcções por parte de outros *developers*, proceder a diferenças nas diferentes revisões no repositório e automatizar tarefas como o envio de e-mails, *builds* automáticos de sistema e geração de arquivos com os binários de *runtime* (com o *plugin* Bitten).

Por estas razões, foi possível aferir que a primeira metodologia ia mais facilmente ao encontro da ideia inicial da tese: ”*É possível sistematizar e melhorar consideravelmente o ciclo de produção de software, extendendo as funcionalidades de um SCM (por exemplo SVN) recorrendo unicamente a ferramentas open source*”.

The most exciting phrase to hear in science – the one that heralds new discoveries – is not "Eureka!" but "That's funny...".

I. Asimov

5 Conclusões e Trabalho Futuro

Neste capítulo final é feito um resumo de todo o trabalho realizado e descrita a satisfação dos objectivos traçados para a dissertação no Capítulo 1. São também feitas recomendações para o trabalho futuro.

5.1 Satisfação dos Objectivos

A dissertação conseguiu mostrar que é possível aumentar a produtividade de uma equipa de desenvolvimento com a re-estruturação do ciclo de desenvolvimento de software recorrendo unicamente a ferramentas *open source*. De facto, o uso da primeira metodologia conseguiu aumentar a velocidade de correcção de *bugs*, a colaboração entre os *developers*, a preparação de *deliverables* pela equipa de testes e a integração de todas estas funcionalidades num *workflow* baseado na ferramenta Trac. Por estes motivos, a primeira metodologia ganhou um relevo importante em relação à segunda metodologia proposta. Uma dificuldade encontrada foi na obtenção de métricas quantitativas que permitissem concluir precisamente o ganho real na adopção do SVN como principal VCS dentro da equipa de desenvolvimento. Esta dificuldade não tornou impossível de perceber empiricamente se o uso do SVN estaria a dar frutos, mas foi necessário recolher métricas qualitativas para verificar isto mesmo. As métricas qualitativas usadas foram a facilidade de uso do ambiente Trac para realizar as tarefas mais habituais (ver o estado do código-fonte, que ramos

existem, quem alterou o quê, quando e porquê), o uso do SVN pela linha de comandos e através do Eclipse, e por fim, a interligação do *plugin* Mylyn no Eclipse com o consequente ganho na produtividade. Estes indicadores permitiram aferir rapidamente que a adopção do SVN foi uma aposta de sucesso em que toda a equipa beneficiou deste facto.

5.2 Trabalho Futuro

Como trabalho futuro, uma possibilidade é a extensão do ambiente Trac para subsitituir por completo o *Project Link* e também para aumentar as suas funcionalidades com a geração automática de documentos em formato PDF. A injecção das horas de trabalho nas *timesheets* Excel através do *plugin* Mylyn é outro dos pontos cruciais para o trabalho futuro que tornaria ainda mais fácil o ciclo de desenvolvimento de software para as equipas de desenvolvimento e de teste.
Por fim, o uso destas metodologias às restantes equipas de desenvolvimento das diferentes divisões na EFACEC seria um objectivo a curto prazo extremamente importante. Para além de aumentar a produtividade nos processos de desenvolvimento, os diferentes produtos estariam integrados sob as mesmas ferramentas *open source* com a clara vantagem de ser fácil de instalar, configurar e actualizar (feito uma vez, faz-se *n* vezes...), de possuir custos muito baixos nesta solução, e de ter uma curva rápida de aprendizagem, como foi possível de constatar com a equipa de desenvolvimento e de testes do departamento de I&D da divisão de Gestão de Redes.

Referências

[AB09] MySQL AB. MySQL, Fevereiro 2009. Disponível em http://mysql.com, acedido a última vez em 10 de Fevereiro de 2009.

[aP09] Portugal a Programar.org. Portugal-a-Programar.org, A comunidade Portuguesa de Programadores, Fevereiro 2009. Disponível em http://www.portugal-a-programar.org/forum/index.php, acedido a última vez em 26 de Fevereiro de 2009.

[BCS08] Brian W. Fitzpatrick e C. Michael Pilato Ben Collins-Sussman. *Version Control with Subversion*. O'Reilly Media, Second edition, 2008.

[Bor09] Borland. Borland StarTeam - A Complete Software Change & Configuration Management (SCM) Tool, Março 2009. Disponível em http://www.borland.com/us/products/starteam/index.html, acedido a última vez em 1 de Março de 2009.

[Can09] Canonical. Ubuntu 8.10 Desktop Edition, Janeiro 2009. Disponível em http://www.ubuntu.com/products/whatisubuntu/810features/, acedido a última vez em 12 de Janeiro de 2009.

[Col09a] CollabNet. Subclipse – SVN plug-in for Eclipse, Fevereiro 2009. Disponível em http://subclipse.tigris.org, acedido a última vez em 10 de Fevereiro de 2009.

[Col09b] CollabNet. Subversion, Janeiro 2009. Disponível em http://subversion.tigris.org, acedido a última vez em 12 de Janeiro de 2009.

[Col09c] CollabNet. Subversion License, Janeiro 2009. Disponível em http://subversion.tigris.org/license-1.html, acedido a última vez em 12 de Janeiro de 2009.

[Col09d] CollabNet. Subversion Mailing Lists, Fevereiro 2009. Disponível em http://subversion.tigris.org/mailing-lists.html, acedido a última vez em 10 de Fevereiro de 2009.

[Col09e] CollabNet. TortoiseSVN, Fevereiro 2009. Disponível em http://tortoisesvn.tigris.org/, acedido a última vez em 10 de Fevereiro de 2009.

[Col09f] CollabNet. TortoiseSVN Downloads, Fevereiro 2009. Disponível em http://tortoisesvn.net/downloads, acedido a última vez em 10 de Fevereiro de 2009.

[Col09g] CollabNet. Where Subversion Meets the Enterprise, Janeiro 2009. Disponível em http://www.collab.net/, acedido a última vez em 18 de Janeiro de 2009.

[Com09] Free Software Community. Eclipse IDE, Fevereiro 2009. Disponível em http://www.eclipse.org, acedido a última vez em 10 de Fevereiro de 2009.

[Cor09a] Microsoft Corporation. MSN Live Messenger, Janeiro 2009. Disponível em http://messenger.live.com/, acedido a última vez em 15 de Janeiro de 2009.

[Cor09b] Microsoft Corporation. Visual SourceSafe, Janeiro 2009. Disponível em http://msdn.microsoft.com/en-us/vstudio/aa700907.aspx, acedido a última vez em 18 de Janeiro de 2009.

[Cor09c] Microsoft Corporation. Visual Studio 2005 Team Foundation Server, Janeiro 2009. Disponível em http://msdn.microsoft.com/en-us/vs2005/aa718825.aspx, acedido a última vez em 18 de Janeiro de 2009.

[Cor09d] Microsoft Corporation. Visual Studio Developer Center, Março 2009. Disponível em http://msdn.microsoft.com/vstudio/, acedido a última vez em 1 de Março de 2009.

[Cor09e] Microsoft Corporation. Windows, Janeiro 2009. Disponível em http://windows.microsoft.com, acedido a última vez em 12 de Janeiro de 2009.

[EFA09a] EFACEC. Grupo EFACEC – EFACEC no Mundo, Fevereiro 2009. Disponível em http://www.efacec.pt/PresentationLayer/efacec_ctexto_00.aspx?idioma=1&local=6&area=1, acedido a última vez em 26 de Fevereiro de 2009.

[EFA09b] EFACEC. Grupo EFACEC – Estrutura Societária, Fevereiro 2009. Disponível em http://www.efacec.pt/PresentationLayer/efacec_ctexto_00.aspx?idioma=1&local=5&area=1, acedido a última vez em 26 de Fevereiro de 2009.

[EFA09c] EFACEC. Grupo EFACEC – Unidades Fabris, Fevereiro 2009. Disponível em http://www.efacec.pt/PresentationLayer/efacec_ctexto_00.aspx?idioma=1&local=11&area=1, acedido a última vez em 26 de Fevereiro de 2009.

[Fou09a] Apache Software Foundation. Apache HTTP Server, Fevereiro 2009. Disponível em http://httpd.apache.org, acedido a última vez em 10 de Fevereiro de 2009.

[Fou09b] Eclipse Foundation. Eclipse Communication Framework, Março 2009. Disponível em http://www.eclipse.org/ecf/, acedido a última vez em 1 de Março de 2009.

[Fou09c] Python Software Foundation. Python Programming Language, Fevereiro 2009. Disponível em http://www.python.org, acedido a última vez em 10 de Fevereiro de 2009.

[Goo09a] Google. Google Talk, Janeiro 2009. Disponível em http://www.google.com/talk/, acedido a última vez em 15 de Janeiro de 2009.

[Goo09b] Google. NautilusSVN, Março 2009. Disponível em http://code.google.com/p/nautilussvn/, acedido a última vez em 1 de Março de 2009.

[Gro09] PostgreSQL Global Development Group. PostgreSQL, Fevereiro 2009. Disponível em http://www.postgresql.org, acedido a última vez em 10 de Fevereiro de 2009.

[Hip09] D. Richard Hipp. SQLite, Fevereiro 2009. Disponível em http://sqlite.org, acedido a última vez em 10 de Fevereiro de 2009.

[IBM09] IBM. Rational ClearCase, Janeiro 2009. Disponível em http://www-01.ibm.com/software/awdtools/clearcase, acedido a última vez em 18 de Janeiro de 2009.

[Inc09] Apple Inc. XCode, Março 2009. Disponível em http://developer.apple.com/tools/xcode/, acedido a última vez em 1 de Março de 2009.

[Jet09] JetBrains. IntelliJIDEA, Março 2009. Disponível em http://www.jetbrains.com/idea/, acedido a última vez em 1 de Março de 2009.

[Ker09] Mik Kersten. Mylyn, Março 2009. Disponível em http://www.eclipse.org/mylyn/, acedido a última vez em 1 de Março de 2009.

[Lim09] Skype Limited. Skype, Janeiro 2009. Disponível em http://www.skype.com/, acedido a última vez em 15 de Janeiro de 2009.

[LLC09] AOL LLC. ICQ, Janeiro 2009. Disponível em http://www.icq.com/, acedido a última vez em 15 de Janeiro de 2009.

[Mic09] Sun Microsystems. NetBeans IDE, Março 2009. Disponível em http://www.netbeans.org/, acedido a última vez em 1 de Março de 2009.

[MR09] Paulo Santos Miguel Rentes. SCATEX Configuration Management, Março 2009. EFACEC Engenharia.

[Nag05] William A. Nagel. *Subversion Version Control: Using The Subversion Version Control System in Development Projects*. Prentice Hall, First edition, 2005.

[Pin04] Vasco Moreira Pinto. *Interligação BUS/JFRK na EFACEC Sistemas de Electrónica, SA*. Relatório do Estágio Curricular LEIC, Faculdade de Engenharia da Universidade do Porto, 2004.

[Pri09] Derek Robert Price. CVS - Concurrent Versions System, Janeiro 2009. Disponível em http://www.nongnu.org/cvs/, acedido a última vez em 18 de Janeiro de 2009.

[Sof09a] Edgewall Software. Bitten - A continuous integration plugin for Trac, Março 2009. Disponível em http://bitten.edgewall.org/, acedido a última vez em 1 de Março de 2009.

[Sof09b] Edgewall Software. Genshi Template Engine, Fevereiro 2009. Disponível em http://genshi.edgewall.org, acedido a última vez em 10 de Fevereiro de 2009.

[Sof09c] Edgewall Software. Trac Integrated SCM & Project Management, Janeiro 2009. Disponível em http://trac.edgewall.org/, acedido a última vez em 12 de Janeiro de 2009.

[Sof09d] Edgewall Software. Trac Plugin List, Fevereiro 2009. Disponível em http://trac.edgewall.org/wiki/PluginList, acedido a última vez em 10 de Fevereiro de 2009.

[Sof09e] TMate Software. SVNKit - SubVersioning for Java, Março 2009. Disponível em http://svnkit.com/, acedido a última vez em 1 de Março de 2009.

[Som06] Ian Sommerville. *Software Engineering*. Addison Wesley, Eight edition, 2006.

[Wik09a] Wikipedia. Agile Software Development, Janeiro 2009. Disponível em http://en.wikipedia.org/wiki/Agile_software_development, acedido a última vez em 12 de Janeiro de 2009.

[Wik09b] Wikipedia. Comparison of revision control software, Janeiro 2009. Disponível em http://en.wikipedia.org/wiki/Comparison_of_revision_control_software, acedido a última vez em 12 de Janeiro de 2009.

[Wik09c] Wikipedia. Coordinated Universal Time, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/UTC, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09d] Wikipedia. DOS, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/DOS, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09e] Wikipedia. Extreme Programming, Janeiro 2009. Disponível em http://en.wikipedia.org/wiki/Extreme_programming, acedido a última vez em 12 de Janeiro de 2009.

[Wik09f] Wikipedia. Instant Messaging, Janeiro 2009. Disponível em http://en.wikipedia.org/wiki/Instant_messaging, acedido a última vez em 12 de Janeiro de 2009.

[Wik09g] Wikipedia. International Organization for Standardization, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/ISO, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09h] Wikipedia. Linux, Janeiro 2009. Disponível em http://en.wikipedia.org/wiki/Linux, acedido a última vez em 26 de Fevereiro de 2009.

[Wik09i] Wikipedia. MD5, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/MD5, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09j] Wikipedia. Microsoft Visual SourceSafe Criticism, Janeiro 2009. Disponível em http://en.wikipedia.org/wiki/Visual_SourceSafe#Criticisms, acedido a última vez em 18 de Janeiro de 2009.

[Wik09k] Wikipedia. MIME, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/MIME, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09l] Wikipedia. Newline, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/CRLF, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09m] Wikipedia. Peer-to-Peer, Março 2009. Disponível em http://en.wikipedia.org/wiki/Peer-to-peer, acedido a última vez em 1 de Março de 2009.

[Wik09n] Wikipedia. Revision Control, Janeiro 2009. Disponível em http://en.wikipedia.org/wiki/Source_Code_Management, acedido a última vez em 12 de Janeiro de 2009.

[Wik09o] Wikipedia. Secure Shell, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/Secure_Shell, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09p] Wikipedia. Secure Sockets Layer, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/Transport_Layer_Security, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09q] Wikipedia. Uniform Resource Locator, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/URL, acedido a última vez em 10 de Fevereiro de 2009.

[Wik09r] Wikipedia. Unix, Janeiro 2009. Disponível em http://en.wikipedia.org/wiki/Unix, acedido a última vez em 26 de Fevereiro de 2009.

[Wik09s] Wikipedia. Web-based Distributed Authoring and Versioning, Fevereiro 2009. Disponível em http://en.wikipedia.org/wiki/WebDAV, acedido a última vez em 10 de Fevereiro de 2009.

He [John von Neumann] had the invaluable faculty of being able to take the most difficult problem and separate it into its components, whereupon everything looked brilliantly simple.

S. Ulam

A

Introdução ao Subversion

O controlo de versões de um *software* sempre foi uma questão muito importante e delicada para qualquer programador ou equipa de programadores de *software*. É actualmente impensável desenvolver projectos de *software* complexos e com requisitos que estão frequentemente a mudar, sem ter um sistema que controle tudo o que acontece a esse mesmo *software*: as alterações ao código, a possibilidade de voltar sempre atrás até uma versão estável do *software* (óptimo quando ocorrem erros e se precisa de regredir no código), ver que alterações foram feitas no código, globalmente ou apenas numa porção deste (incluindo as respostas às perguntas: *Quando? Por quem? Para que fim? Em que ficheiros?*), controlar eficazmente a fusão (*merge*) das alterações em vários módulos separados de *software* num mesmo produto unificado, e também controlar a própria documentação.

Sem dúvida que em qualquer projecto de *software* (por muito pequeno ou muito grande que seja), o controlo da informação é crucial para o sucesso do mesmo. Por isto mesmo, a tarefa de controlar as versões de um *software* não é de todo fácil, nem todos os sistemas de controlo de versões, ou **VCS** - *Version Control System* [Wik09n], são iguais e com as mesmas funcionalidades [Wik09b].

Neste anexo vamos ver como funciona o Subversion [Col09b]: um sistema de controlo de versões open source (*software* livre) de enorme sucesso actual. Vamos ver também como se instala o Subversion (cliente e servidor) para as plataformas Linux Ubuntu [Can09] e Windows [Cor09e], como se configura o servidor HTTP da Apache (usufruindo logo à partida das vantagens da autenticação e autorização) para usar URLs dos repositórios SVN, como se usa o Subversion no dia-a-dia do trabalho de um programador e por fim, que ferramentas gráficas ou *IDE*'s (*Integrated Development Environment*) existem à disposição de todos para serem usadas com o Subversion.

No final da leitura deste anexo, espera-se que o leitor fique a conhecer e a saber usar o Subversion como uma ferramenta de trabalho muito útil para qualquer programador (e mesmo não-programador) no que toca à gestão das alterações do *software*, ou seja, controlando totalmente o percurso do *software*.

A.1 Porquê controlar o que acontece ao *software*?

As funcionalidades básicas de qualquer sistema de controlo de versões são o de permitir anotar todas as alterações dos ficheiros ao longo do tempo e de unir as contribuições de vários programadores. Para suportar isto, o VCS tem de manter um histórico de todas as alterações que foram feitas ao longo do tempo por diferentes pessoas. Desta forma, é possível voltar atrás no código até uma revisão mais antiga, e ver que aspecto tinham os ficheiros nessa altura. Para além disso, o VCS tem de garantir sempre a correcta integração das alterações feitas no código. Mas o que se ganha efectivamente com o controlo das versões de *software*? Especialmente, para uma equipa pequena que benefícios se podem tirar de um bom VCS que valham a pena perder tempo a aprender como usá-lo no dia-a-dia? Eis algumas das razões para usar sempre um VCS [Nag05]:

1. **Integridade dos dados** – Um bom VCS ajuda a proteger a integridade dos dados. Ao manter um histórico das revisões (que são os conjuntos atómicos de alterações) deixa de haver a preocupação de possivelmente se estar a apagar código que mais tarde vamos perder tempo a reproduzir e que afinal era mesmo importante. Com um bom VCS podemos voltar sempre a qualquer snapshot do código, inclusivamente a esse momento anterior a termos apagado código. Esta característica também é útil no backup de dados, porque se os programadores guardarem regularmente as suas alterações no repositório central (no SVN, isto designa-se de commit) que tem todo o código, então é fácil ter um processo automático que arquive todo este código do repositório, e assim resolve-se o problema do código não estar unicamente no computador de um programador (mesmo à espera que aconteça aquela falha inevitável no disco...).

2. **Produtividade** – Ao libertar os programadores da tarefa árdua de ter que integrar manualmente todas as alterações que fizeram (e tipicamente em muitos ficheiros ao mesmo tempo), um VCS pode aumentar bastante a productividade na criação de *software*. De facto, com um bom VCS os programadores conseguem testar as alterações que fizeram e que outros programadores fizeram, resolver possíveis conflitos de código (que correspondem a mudanças nos mesmos ficheiros e nas mesmas linhas desses ficheiros) mesmo antes de incluir esse código no repositório central, e de uma forma automática. E, com mais tempo para programar, nasce melhor *software* (se o programador não se distraír das suas programações).

3. **Gestão de Projectos** – Um bom VCS consegue reunir rapidamente um conjunto de informações muito úteis para um gestor de projectos: quem alterou o quê, quando, porquê, quantas modificações fez no mesmo projecto (ou módulo), e se as alterações feitas vão de encontro à finalidade do projecto (por exemplo, se existirem várias modificações que não resolvem os bugs encontrados). Estas informações são importantes para os gestores de projectos e para os programadores que não são apenas programadores mas também líderes de pequenas equipas de *software*. Apesar de um VCS estar principalmente preocupado com a gestão das versões, as informações que fornece podem ser facilmente usadas por *software* mais específico da área de gestão de projectos.

4. **Suporte a projectos de Engenharia de *software*** – Tipicamente, bons projectos de *software* são desenvolvidos com um processo de engenharia de *software*. Por engenharia de *software*, quero dizer a aplicação de políticas de desenvolvimento com o intuito de assegurar que o producto final do processo vai de encontro aos objectivos, dentro dos prazos previstos e com os melhores padrões possíveis de qualidade do *software*. Um bom processo de engenharia de *software* envolve um conjunto de passos, tais como o desenho bem detalhado do projecto na sua globalidade, a revisão dos componentes do *software* por parte dos analistas e engenheiros, a anotação de todos os bugs, e testes de qualidade ao *software*. Nenhum destes passos é explicitamente suportado por qualquer VCS actualmente, mas as características de muitos VCS (como por exemplo, os *hook scripts* e *logs* do SVN) são excelentes ajudas ao processo de engenharia de *software*. Vamos ver mais adiante neste anexo o que o SVN nos fornece neste campo.

5. **Ramificação do desenvolvimento** – À medida que os projectos vão aumentando, há necessidade de ramificar o projecto. Isto é, vai surgir naturalmente a necessidade de um programador (ou conjunto de programadores) desenvolver features que não podem de maneira nenhuma minar o trabalho que já existe guardado num repositório de código central. Por exemplo, basta pensar num novo módulo que vai demorar tempo a ser desenvolvido e que vai ter repercussões nos restantes módulos. Neste caso, o ideal é o programador trabalhar num ramo privado, em que este ramo não é mais do que uma cópia de todo o código que já existia e no qual vai poder alterar à vontade, sem o problema de estar a interferir com o código de outros programadores. No final, quando todas as alterações estiverem feitas, o programador funde todo o seu código com o que existe no repositório central (podendo ter que resolver conflitos), e o ramo pode deixar de existir. Outro caso em que os ramos são importantes é no suporte a versões antigas do *software*. Cada uma das versões antigas poderá ser um ramo do código a dada altura (por exemplo, na altura em que sai uma nova versão), e a equipa de programadores pode testar a resolução dos bugs nesses ramos de código antigo e posteriormente entregar ao cliente uma versão corrigida desse *software* para aquela versão anterior. Nem todos os VCS possuem esta funcionalidade nem tão pouco fornecem uma alternativa. No entanto, o VCS analisado neste artigo fornece esta a outras funcionalidades importantes para o controlo do *software*.

6. **Anotação dos logs** – Para além de saber quem alterou o quê, é também importante saber o porquê. A cada alteração deverá ficar sempre associada uma mensagem que traduza a alteração que se fez e que pode posteriormente ser usada para construir um `CHANGELOG` ou ainda ser injectada num sistema de tracking, tal como o Trac [Sof09c] de que falaremos mais abaixo.

7. **Distribuição de trabalho** – Actualmente, com a era da globalização e do alcance fácil de uma ligação à Internet, o trabalho à distância é uma realidade presente em cada vez mais empresas, seja o programador que está a aceder ao código do outro lado da cidade ou seja uma empresa sub-contratada do outro lado do planeta; um VCS deverá conseguir equilibrar esta distribuição de trabalho ao gerir todas estas alterações ao código pelos diferentes programadores e também unificar este mesmo código automaticamente. Combinando estas características com uma comunicação

(que é fundamental) através de e-mail ou um cliente de IM [Wik09f] (como o GTalk [Goo09a], MSN Messenger [Cor09a], Skype [Lim09] ou ICQ [LLC09]), é praticamente tão fácil de trabalhar remotamente como localmente ao lado dos restantes elementos da equipa.

8. **Desenvolvimento rápido** – As mais recentes metodologias de desenvolvimento de *software* apontam para uma forma de programar mais rápida, e flexível usando metodologias de *Extreme Programming* (XP) [Wik09e] e de *Desenvolvimento Ágil* de *software* [Wik09a]. Estas metodologias de desenvolver *software*, esperam por iterações mais rápidas e sucessivas relativamente a metodologias mais "planeadas". Esta forma de programar é facilitada por um VCS que consiga manter e indicar todas as alterações por que passou o *software* nas sucessivas iterações. Para além disso, um repositório central de *software* pode ajudar na automatização dos *builds* frequentes que são precisos por um processo de desenvolvimento ágil.

A.2 O que é o Subversion?

O Subversion [Col09b] é um VCS *open source* que tem tido enorme sucesso mundial e tem sido escolhido não só para a gestão de pequenos projectos *open source*, como também para projectos de elevada complexidade por parte de grandes empresas actuais de *software* (Google, Mozilla, entre outras). É um VCS livre, a custo zero, que se pode modificar se necessário [Col09c], e tem-se revelado como a melhor alternativa actual a VCS comerciais, como é o caso do Rational ClearCase da IBM [IBM09], o Microsoft SourceSafe [Cor09b] (que apresenta já globalmente algum criticismo [Wik09j]; uma melhor alternativa actual é o Microsoft Team Foundation Server [Cor09c]) ou ainda o Borland StarTeam [Bor09].

É um sistema de controlo de versões que faz a gestão de ficheiros e directórios ao longo do tempo. Surgiu em 2000 pelas mãos da empresa CollabNet [Col09g] com o objectivo principal de substituir o CVS – *Concurrent Versions System* [Pri09] – que até então era bastante usado, mas que tinha claras limitações na usabilidade.

Um dos conceitos chave do SVN é o de ter uma árvore de ficheiros colocados num repositório central. O repositório é como um servidor de ficheiros, com a excepção de que se *"lembra"* de todas as mudanças feitas nos ficheiros e directórios. Isto permite que versões antigas de *software* sejam recuperadas, ou apenas que se possa examinar o histórico das alterações feitas nos dados. O Subversion pode também aceder a repositórios remotos, o que permite que possa ser utilizado por pessoas em diferentes máquinas, permitindo assim que haja uma maior colaboração no desenvolvimento de *software*. Este desenvolvimento pode, portanto, ser mais rápido, e uma vez que há versões no código desenvolvido, não há receio de que se perca qualidade no *software* uma vez que se houver uma mudança incorrecta no *software*, basta apenas retroceder nessa alteração. Simples e rápido.

O Subversion fornece:

- **Versionamento de directórios** – o CVS mostra apenas o histórico de ficheiros individuais, mas o Subversion implementa um sistema "virtual"de versões que pode tomar nota das alterações a árvores inteiras de directórios ao longo do tempo. Desta maneira, ficheiros e directorias têm versões;

- **Histórico de versões** – uma vez que o CVS é limitado nas versões dos ficheiros, operações como copiar ou renomear ficheiros não são suportadas. O CVS também não consegue substituir um ficheiro com uma determinada versão por outro ficheiro com o mesmo nome, sem que antes esse novo ficheiro tenha herdado todo o histórico do ficheiro antigo, e pode acontecer que este novo ficheiro não tenha nenhuma relação com o antigo do mesmo nome. Com o Subversion pode-se adicionar, apagar, copiar, e renomear tanto ficheiros como directorias. E cada novo ficheiro adicionado começa com um histório limpo apenas para aquele ficheiro;

- **Commit's atómicos** – um conjunto de modificações enviadas para o servidor de SVN (designado de *commit*) vai para o repositório de uma só vez como um conjunto global, ou não vai de todo (precisamente como um commit em SQL, isto é, se o commit falhar durante a operação, nada é passado para o servidor). Isto permite que os programadores possam construir e aplicar as modificações ao *software* como conjuntos lógicos de modificações de *software*, prevenindo que possam ocorrer problemas quando apenas uma parte do conjunto de alterações é enviado com sucesso para o repositório;

- **Versionamento de meta-dados** – cada ficheiro e directoria tem um conjunto de propriedades - chaves e os valores das chaves - associadas. Pode-se criar e guardar qualquer par arbitrário chave/valor, que se deseje. As propriedades têm também versões ao longo do tempo, como para os conteúdos dos ficheiros; As propriedades fornecem um maior controlo e customização do código-fonte no repositório;

- **Escolha de redes de acesso** – o Subversion tem uma noção abstracta de acesso a um repositório, tornando mais fácil a criação de novos mecanismos de rede. O Subversion pode ligar-se ao servidor de HTTP da Apache como um módulo de extensão. Isto dá ao Subversion uma grande vantagem na estabilidade e interoperabilidade, e acesso instantâneo às funcionalidades existentes fornecidas por este servidor HTTP - autenticação, autorização, etc. Uma versão mais leve, sem se ligar ao servidor HTTP do Apache, do servidor de Subversion está também disponível (designada por `svnserve`). Este servidor – `svnserve` – comunica através de um protocolo que pode ser facilmente usado num túnel SSH;

- **Tratamento consistente dos dados** – o Subversion percebe as diferenças nos ficheiros através de um algoritmo que diferencia os binários de texto, e que funciona tão bem para ficheiros de texto como para ficheiros binários. Ambos os tipos de ficheiros são comprimidos no repositório, e as diferenças são transmitidas em ambas as direcções através da rede de computadores;

- **Ramificação (*branching*) e Etiquetagem (*tagging*) eficientes** – o custo de ramificar e etiquetar não são proporcionais ao tamanho do projecto. O Subversion cria ramos e etiquetas ao copiar o projecto usando um mecanismo semelhante a um link em Linux. Por isso, estas operações levam um período de tempo bastante curto e constante. O que é excelente.

- **Usabilidade** – o Subversion é implementado numa colecção de bibliotecas C partilhadas com API's bem definidas. Isto torna o Subversion extremamente fácil de manter e modificar por outras aplicações e linguagens de programação. Existe

também actualmente uma biblioteca Java, SVNKit, que implementa o Subversion e que pode ser usada por qualquer aplicação Java.

A.3 Arquitectura do SVN

Na Figura A.1 indica-se a arquitectura do Subversion numa visão de alto-nível.

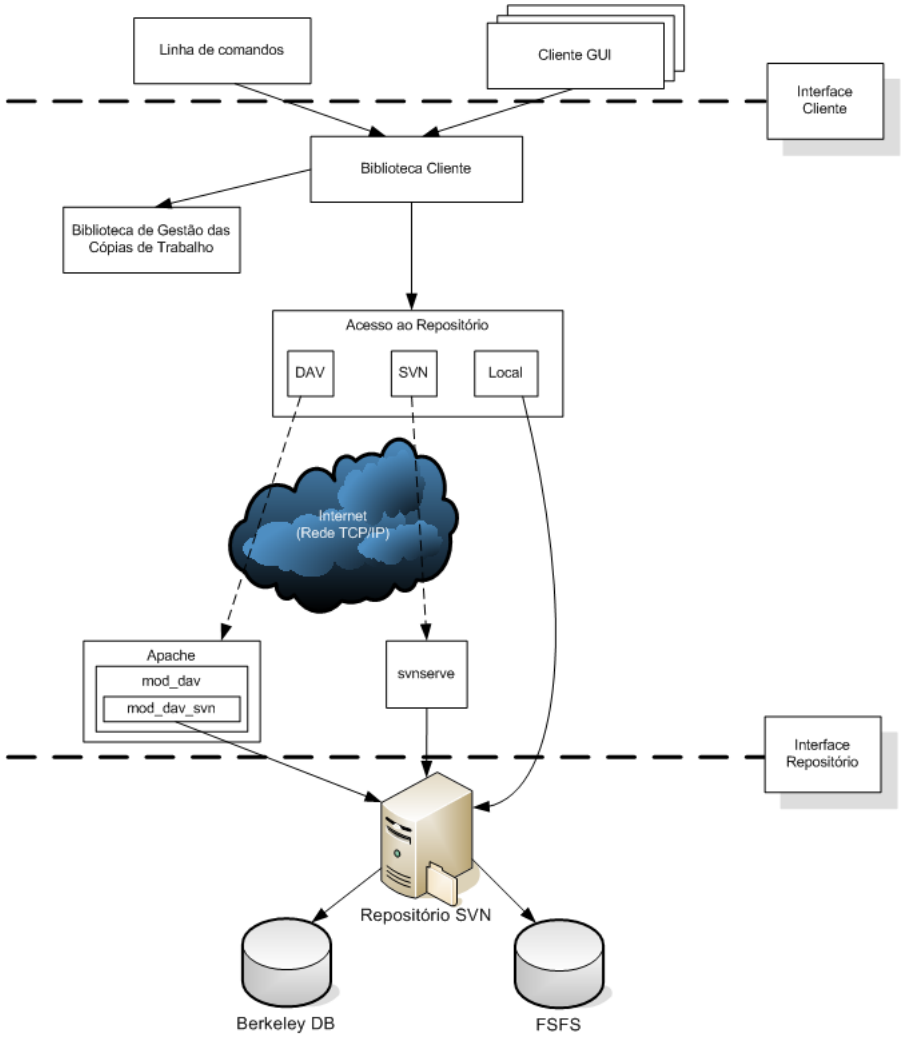

Figura A.1: Arquitectura do Subversion.

Como se pode ver, o Subversion segue uma filosofia de implementação do tipo cliente-servidor. Numa extremidade está o repositório do Subversion que tem o projecto com as diferentes versões. Na outra extremidade está o cliente de Subversion, que gere as porções locais do projecto em determinada versão (chamadas cópias de trabalho – *working copies*). Entre estas extremidades há vários caminhos possíveis através das várias camadas de acesso ao repositório (RA – *Repository Access*). Alguns destes caminhos (ou percursos), atravessam redes de computadores e terminam em servidores que acedem ao repositório.

Outros caminhos não passam por redes de computadores e levam a um acesso directo ao repositório. Existem dois tipos de filesystem para os repositórios SVN: *FSFS* (o mais

recente e recomendado) e o *Berkeley DB* da Oracle. Com as versões mais recentes do SVN (desde a versão 1.4.6), o *filesystem* criado por omissão é *FSFS*.

A.4 Conceitos básicos do Subversion

O Subversion, tal como foi dito acima, é um sistema centralizado de partilha de informação. No seu núcleo estrutural encontra-se um repositório central de dados. Este repositório guarda a informação sob a forma de uma árvore de um sistema de ficheiros - tal como uma hierarquia típica de ficheiros e directorias. Qualquer número de clientes ligam-se ao repositório (directa ou indirectamente), e lêem ou escrevem para os ficheiros do repositório. Ao escrever, o cliente torna a informação disponível a outros utilizadores; ao ler, o cliente recebe informação de outros utilizadores. Recorde-se que o Subversion não permite apenas partilhar código-fonte de qualquer aplicação de *software*, mas sim todo o tipo de ficheiros (texto ou binários).

Apesar de parecer funcionar exactamente do mesmo modo que um normal servidor de ficheiros, o Subversion diferencia-se no facto de poder recordar cada alteração feita nos ficheiros, e cada alteração feita na árvore de uma directoria, como por exemplo, a adição, a remoção ou o rearranjo de ficheiros e directorias.

Quando um cliente lê dados do repositório, ele vê normalmente apenas a última versão da árvore do sistema de ficheiros. Porém, o cliente pode ainda ver os estados anteriores do sistema de ficheiros. Por exemplo, um cliente pode colocar questões sobre o histórico tais como: *Que ficheiros tinha esta directoria na passada Quarta-feira?* ou *Quem foi a última pessoa a modificar este ou aquele ficheiro?*, ou até *Que alterações é que essa pessoa fez?*. Este tipo de questões são fundamentais para qualquer sistema de controlo de versões: sistemas desenhados para guardar e tomar nota das alterações aos dados ao longo do tempo.

A.5 O problema do controlo de versões

O objectivo de um sistema de controlo de versões é o de permitir a colaboração na edição e partilha de dados. Isto coloca um problema fundamental: como é que esse sistema permite que os utilizadores partilhem a informação, mas impeça que um utilizador altere acidentalmente um ficheiro que está a ser modificado por outro? Num repositório com um grande número de alterações a serem feitas diariamente, esta situação acontece com grande facilidade.

Consideremos a Figura A.2 em que temos dois programadores fictícios, o Miguel e a Eunice, e decidem editar o mesmo ficheiro do respositório ao mesmo tempo. Se o Miguel guardar as suas alterações no repositório primeiro, então é possível que (passados uns minutos ou umas horas) a Eunice possa acidentalmente *"passar por cima"* das alterações do Miguel na sua própria cópia do ficheiro. Embora as alterações feitas pelo Miguel não sejam perdidas (porque o sistema de versões lembra-se de cada alteração feita), estas não estarão presentes na nova versão do ficheiro criada pela Eunice, pois ela nunca viu as alterações do Miguel. O trabalho do Miguel acaba por ser perdido efectivamente – ou pelo menos não aparece na última versão do ficheiro – e portanto, o sistema de controlo de versões tem de evitar este tipo de problemas.

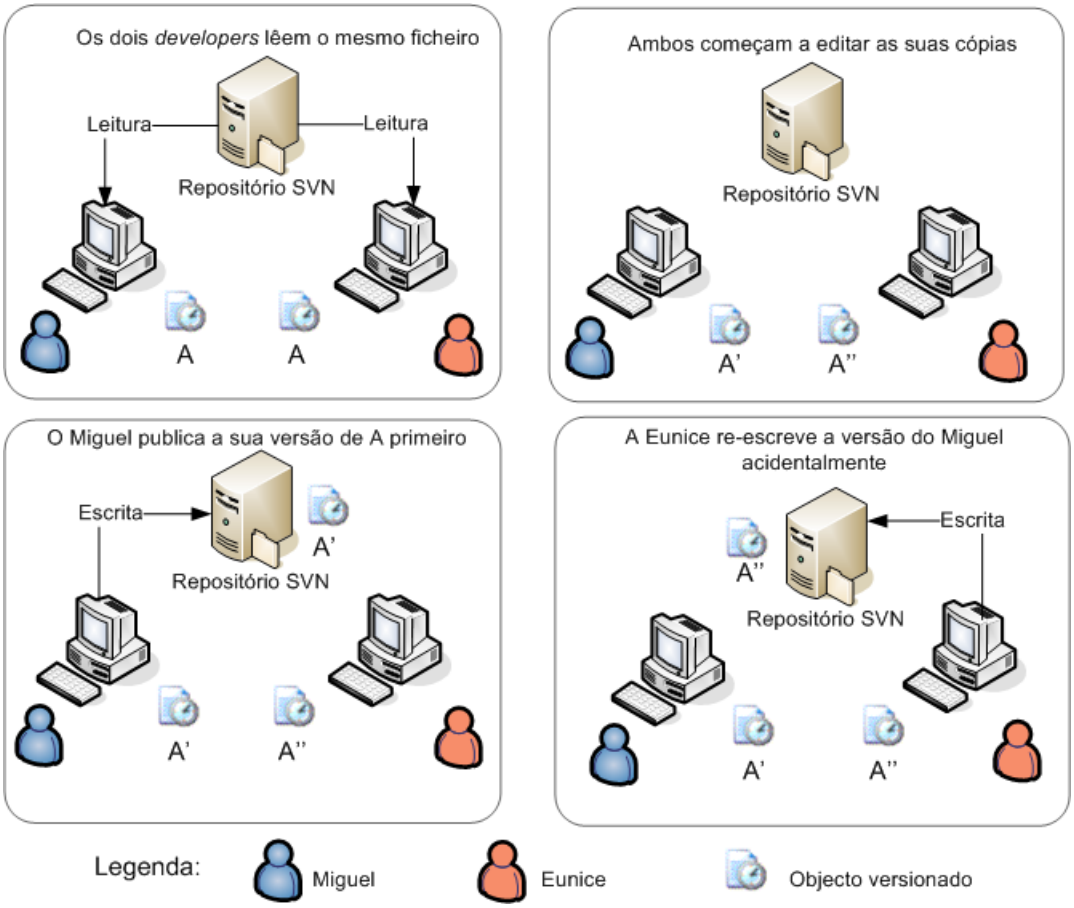

Figura A.2: Problema na escrita concorrente do mesmo ficheiro.

A.6 A solução *Bloqueia-Modifica-Desbloqueia* (*Lock-Modify-Unlock*)

Muitos sistemas de controlo de versões (como por exemplo o SourceSafe da Microsoft) usam o modelo de bloqueia-modifica-desbloqueia para evitar o problema de haver mais do que um utilizador a modificar o mesmo ficheiro no repositório. Neste modelo, o repositório permite que só uma pessoa possa modificar um ficheiro em determinada altura. Esta política é gerida através de bloqueios (*locks*). O Miguel tem que bloquear um ficheiro antes de começar a fazer alterações nele. Se o Miguel bloquear um ficheiro, então a Eunice não pode bloqueá-lo, e portanto não pode fazer nenhuma modificação nesse ficheiro. O que ela pode apenas fazer é ler o ficheiro, e esperar que o Miguel acabe as suas alterações e desbloqueie o ficheiro. Após o Miguel desbloquear o ficheiro, a Eunice pode por sua vez bloquear e editar o mesmo ficheiro.

O problema com este modelo é que é muito restritivo, e muitas vezes torna-se uma dor de cabeça para os utilizadores, por enúmeras razões:

1. **Bloquear pode causar problemas de administração.** Por vezes o Miguel pode bloquear um ficheiro e depois esquecer-se dele. Entretanto, e porque a Eunice está à espera para editar o ficheiro, ela fica de mãos atadas à espera que o Miguel faça as alterações ao ficheiro. E é neste momento que o Miguel vai de férias. Perante este

cenário, a Eunice precisa de falar com um administrador para poder desbloquear o ficheiro do Miguel. Esta situação acaba por causar um atraso desnecessário.

2. **Bloquear pode causar serialização desnecessária.** E se o Miguel estiver a editar o início de um ficheiro, e a Eunice quiser editar apenas o fim desse mesmo ficheiro? Estas alterações que o Miguel e a Eunice fizerem, não se sobrepõem e eles poderiam estar facilmente a editar o ficheiro ao mesmo tempo sem causar conflitos no ficheiro, assumindo que as alterações que os dois fizerem sejam propriamente conduzidas. Não há assim necessidade em haver turnos no bloqueio do ficheiro.

3. **Bloquear pode criar uma noção errada de segurança.** Assumindo que o Miguel bloqueia e edita o ficheiro A, enquanto que a Eunice bloqueia e edita ao mesmo tempo o ficheiro B. Mas suponhamos que os ficheiros A e B dependam um do outro, e que mudanças feitas nos dois ficheiros sejam semanticamente incompatíveis. A partir deste momento, os ficheiros A e B não funcionam mais correctamente. O sistema de bloqueios não consegue resolver este problema – e apesar disso forneceu uma noção errada de que havia segurança no manuseio dos ficheiros. É fácil de imaginar para o Miguel e a Eunice que ao bloquearem um ficheiro, cada um deles está a começar uma tarefa independente e segura, e que não seja dada prioridade a uma discussão, antes de tudo, sobre as futuras mudanças incompatíveis que possam causar no repositório.

A.7 A solução *Copia-Modifica-Unifica* (*Copy-Modify-Merge*)

O Subversion, o CVS, e outros sistemas de controlo de versões usam o modelo copia-modifica-unifica como uma alternativa ao bloqueio. Neste modelo, cada utilizador contacta o servidor para aceder ao repositório, e cria uma cópia de toda a estrutura de ficheiros do projecto. Os utilizadores podem então trabalhar em paralelo, modificando as suas cópias privadas. Finalmente, as cópias privadas são unificadas numa só versão final. O sistema de controlo de versões ajuda muitas vezes nesta unificação, mas em último recurso, o utilizador é responsável por fazer com que a unificação ocorra correctamente sem conflitos.

Eis um exemplo. Suponhamos que o Miguel e a Eunice criam as suas próprias cópias de trabalho do mesmo ficheiro, copiadas do repositório central. Trabalham concorrentemente (ao mesmo tempo, em paralelo) e fazem alterações ao mesmo ficheiro A nas suas cópias locais. A Eunice guarda as suas alterações do ficheiro A no repositório antes do Miguel. Quando o Miguel tenta guardar as suas alterações mais tarde, o repositório informa-o que o ficheiro A dele está desactualizado. Noutras palavras, o ficheiro A no repositório foi alterado desde a última cópia que o Miguel fez do respositório. Neste ponto, o Miguel pede então ao cliente de Subversion para unificar quaisquer alterações que tenham havido no repositório com a sua cópia local do ficheiro A, guardando o resultado na sua cópia local. O mais provável é que as alterações da Eunice não se sobreponham às do Miguel; então, desde que o Miguel tenha os dois conjuntos de alterações integrados, ele grava a sua cópia local de novo para o repositório. A Figura A.3 ilustra esta situação.

Mas e se as alterações da Eunice se sobrepõem às do Miguel? Esta situação é designada de conflito, e normalmente não é difícil de resolver.

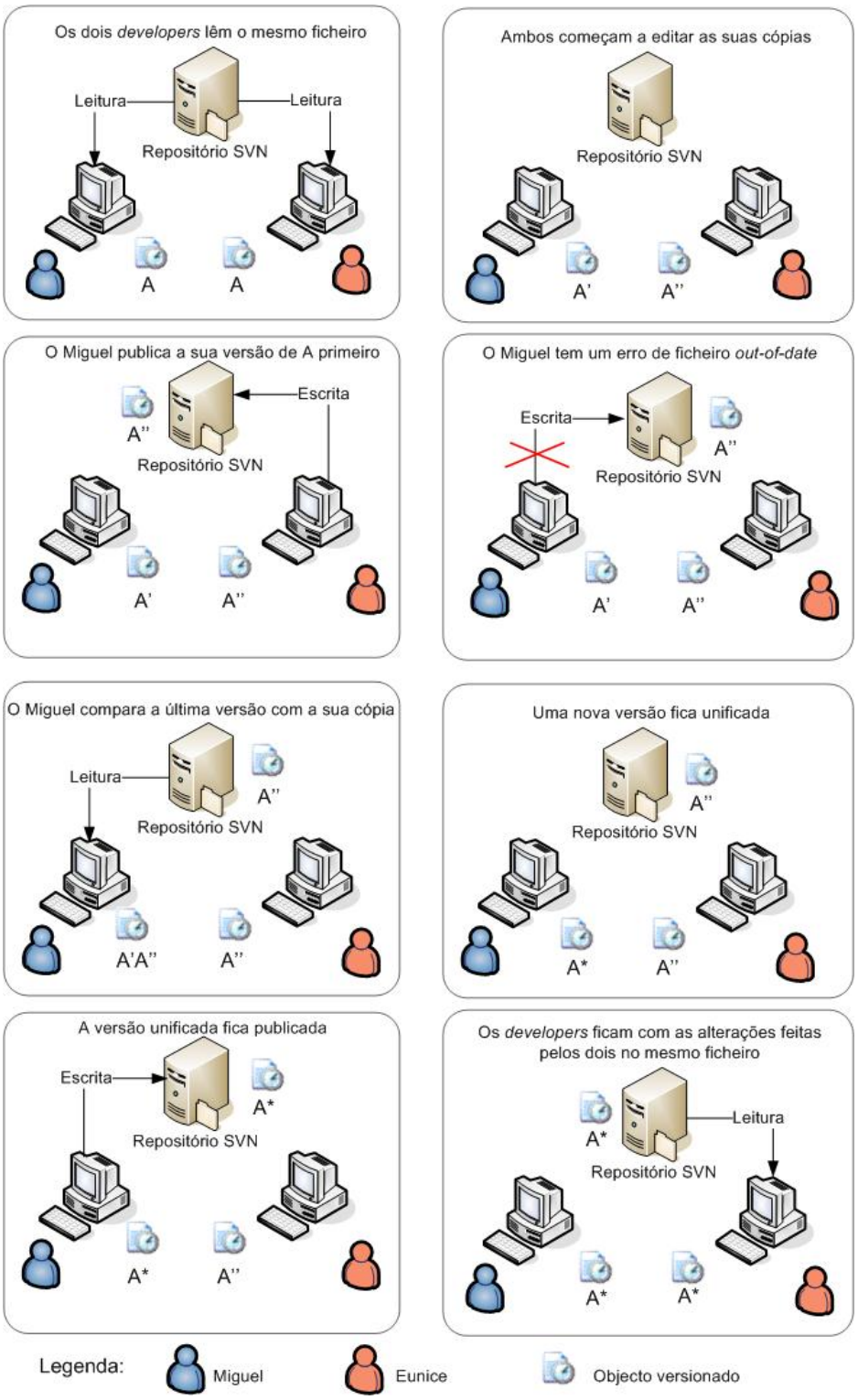

Figura A.3: Alteração concorrente no mesmo ficheiro gerida pelo Subversion.

Quando o Miguel pergunta ao cliente de Subversion para unificar as últimas alterações do repositório com a sua cópia de trabalho, a sua cópia do ficheiro A é alterada para o estado de conflito através de uma flag: ele vai poder ver os dois conjuntos de alterações e poder escolher manualmente entre eles. Note-se que o *software* de controlo de versões não pode por si só resolver os conflitos automaticamente; só os utilizadores é que são capazes de entender e fazer as escolhas certas. A partir do momento em que o Miguel tenha resolvido manualmente os conflitos – após uma conversa com a Eunice, por exemplo – ele pode guardar com segurança o ficheiro unificado de novo no repositório.

O modelo de copia-modifica-unifica pode parecer um pouco caótico, mas na realidade até é bastante simples de usar. Os utilizadores podem trabalhar em paralelo, sem nunca ter que esperar uns pelos outros. Quando se trabalha nos mesmos ficheiros, é interessante verificar que a maioria das alterações que são concorrentes (feitas ao mesmo tempo) não se sobrepõem; os conflitos são pouco frequentes. E o tempo que demora a resolver um conflito é muito menor do que o tempo gasto com um sistema de bloqueios. No centro da questão, tudo converge para um só factor importante: a comunicação entre utilizadores.

Quando os utilizadores comunicam mal, os conflitos aumentam sejam eles sintácticos ou semânticos. Não há nenhum sistema que force a que os utilizadores comuniquem na perfeição, e não há nenhum sistema que consiga detectar conflitos semânticos. Portanto, não adianta pensar que um sistema de bloqueios irá de alguma forma prevenir os conflitos; na prática, os bloqueios inibem a produtividade mais do que qualquer outro modelo de controlo de versões. Apesar disto, pode haver situações que seja de facto útil haver bloqueio de ficheiros. O modelo de copia-modifica- unifica é baseado na ideia de que os ficheiros alterados são passíveis de ser unificados contextualmente: ou seja, a maioria dos ficheiros no repositório são ficheiros de linhas de texto (tal como o código-fonte de um programa). Mas para ficheiros binários, como ficheiros de som ou ficheiros executáveis, é muitas vezes impossível unificar as alterações que geram conflitos. Nestas situações é realmente necessário que os utilizadores tomem a sua vez na alteração do ficheiro. Sem um acesso serializado, alguém acaba sempre por perder tempo com alterações que acabam por ser excluídas. Apesar de o CVS e o Subversion serem essencialmente sistemas do modelo copia-modifica-unifica, ambos reconhecem a necessidade de bloquear um ficheiro ocasionalmente e portanto fornecem mecanismos para o fazer. O bloqueio de ficheiros é suportado pelo Subversion usando a propriedade `svn:needs-lock` ao nível dos ficheiros.

A.8 Revisões

A.8.1 O conceito de Revisão

Um `svn commit` mostra as alterações a qualquer número de ficheiros e directorias numa só transacção atómica. Na cópia de trabalho pessoal de cada utilizador, podem-se alterar os conteúdos dos ficheiros, criar, apagar, renomear e copiar ficheiros e directorias, e só por fim fazer um commit a todo o conjunto de alterações englobando assim todas as alterações como se se tratasse de um só bloco de alterações. No repositório, cada commit é tratado como uma operação atómica: ou todos os commits aos ficheiros acontecem de uma só vez ou nenhum deles acontece. O Subversion tenta manter sempre esta atomicidade para que não aconteçam *crashes*, problemas de rede ou outro tipo de situações que possam prejudicar o *software* desenvolvido e o seu uso correcto.

De cada vez que o repositório aceita um commit, isto cria um novo estado para a árvore do sistema de ficheiros, chamado de revisão. A cada revisão é atribuído um número natural $(1,2,3,...,n)$ único e maior uma unidade do que o número de revisão anterior (incrementa sempre 1 unidade). A revisão inicial de um repositório acabado de criar tem o número de revisão 0, e consiste apenas numa directoria-raíz vazia.

A Figura A.4 ilustra uma forma simples de vermos um repositório. Imagine-se um *array* de números de revisão, começando pelo 0, e aumentando da esquerda para a direita. Cada número de revisão tem uma árvore de ficheiros associada a essa revisão, e cada número de revisão é assim uma maneira de vermos como está um repositório após um *commit*, como se fosse uma fotografia dos ficheiros do repositório.

Figura A.4: A cada conjunto de alterações é agregado e incrementado um novo número de revisão.

A.8.2 Números Globais de Revisão

Os números de revisão do Subversion aplicam-se a árvores inteiras de ficheiros, e não apenas a ficheiros individuais. Cada número de revisão escolhe uma árvore inteira de ficheiros, que é um estado particular do repositório após algumas alterações terem sido aplicadas com o comando commit. Outra forma de se pensar é que a revisão *N* representa o estado dos ficheiros do repositório após o *N*-ésimo `commit`. Quando se fala, por exemplo, na revisão 7 de um ficheiro `ficheiro.c` está-se a referir ao `ficheiro.c` como este aparece na revisão 7. Note-se que em geral, as revisões *N* e *M* de um ficheiro podem não diferir: a revisão é feita globalmente à árvore do sistema de ficheiros e não aos ficheiros individuais; por isso, se um utilizador alterar alguns ficheiros e fizer um `commit`, os ficheiros não alterados passam também a ter um número de revisão superior apesar de não terem sido alterados por esse utilizador. O CVS usa a revisão por ficheiros individuais ao contrário do Subversion e é normal haver alguma dificuldade em usar o Subversion para utilizadores que estão habituados com o CVS.

É importante notar que cópias de trabalho nem sempre correspondem a uma única revisão do repositório; estas podem conter ficheiros de diferentes revisões.

Por exemplo, suponha que se faz checkout de uma cópia de trabalho de um repositório cuja revisão mais recente é a 4:

```
ProjX/Makefile:4
          ficheiroA.c:4
          configure:4
          ...
```

Neste momento, esta cópia de trabalho corresponde exactamente à revisão 4 do repositório. No entanto, suponha que se faz uma alteração ao *ficheiroA.c*, e que se faz *commit* a essa alteração. Assumindo que não foram feitos mais *commits*, vai ser criada a revisão 5 do repositório, e a cópia de trabalho vai parecer-se com isto:

```
ProjX/Makefile:4
          ficheiroA.c:5
          configure:4
          ...
```

Suponha que, a partir deste momento, a Eunice faz *commit* a uma alteração no ficheiro *ficheiroA.c*, criando assim a revisão 6. Se se fizer *svn update* para colocar a cópia de trabalho actualizada, então teremos:

```
ProjX/Makefile:6
          ficheiroA.c:6
          configure:6
          ...
```

A alteração que a Eunice fez no *ficheiroA.c* vai aparecer na cópia de trabalho, e essa mudança vai também estar reflectida nos outros ficheiros que não foram alterados. Neste exemplo, o texto da *Makefile* é o mesmo para as revisões 4, 5 e 6, mas no entanto, o Subversion marca na cópia de trabalho a revisão 6 para a *Makefile* indicando assim que a revisão deste ficheiro ainda é a mais actual. Por isso, após se fazer um *svn update* à cópia de trabalho, isso corresponderá exactamente a uma única revisão no repositório.

A.8.3 Ter revisões diferentes e misturadas num repositório

De cada vez que um utilizador faz *svn commit*, a sua cópia de trabalho acaba por ficar com uma mistura de revisões. As alterações a que se fez um *commit* são marcadas com um número de revisão superior a tudo o resto. Após vários *commit*s (sem *update*s pelo meio) a cópia de trabalho desse utilizador tem uma mistura de revisões diferentes. Mesmo que só haja um utilizador a alterar vários ficheiros, ele vai notar nesta

característica do SVN. Para se ver a mistura de revisões existentes numa cópia de trabalho pode-se usar o comando `svn status -v`.

Frequentemente, novos utilizadores não estão a par de que a sua cópia de trabalho tem diferentes revisões. Isto pode ser confuso porque muitos clientes SVN são sensíveis à revisão de trabalho de um ficheiro que estejam a examinar. Por exemplo, o comando `svn log` é usado para mostrar um histórico das alterações a um ficheiro ou directoria, mas se o número de revisão desse ficheiro ou directoria for bastante antigo (possivelmente porque há já muito tempo que o utilizador não faz um `svn update`), então o histórico da versão mais antiga é que será mostrado ao utilizador.

A.8.4 Revisões misturadas são úteis

Se um projecto é suficientemente complexo, por vezes é útil fazer um retrocesso a algumas porções da cópia de trabalho para uma revisão mais antiga. Às vezes é útil testar uma versão mais antiga de um sub-módulo dentro de uma subdirectoria, ou talvez seja necessário saber quando um *bug* foi criado num ficheiro específico. Este é o aspecto de máquina do tempo de um sistema de controlo de versões - a característica que permite a um utilizador mover qualquer porção da cópia de trabalho para uma revisão mais antiga ou mais recente.

A.8.5 Limitações das revisões misturadas

No entanto, o uso de revisões diferentes e misturadas numa cópia de trabalho tem limitações à sua flexibilidade. Em primeiro lugar, não se pode fazer `commit` quando se apaga um ficheiro ou uma directoria que não esteja com a revisão mais actual. Se uma versão mais recente do ficheiro existe no repositório, a tentativa de o apagar vai ser rejeitada, para prevenir que se apaguem acidentalmente alterações que o utilizador ainda não teve conhecimento. Por fim, não se pode fazer `commit` numa alteração às propriedades de uma directoria (meta-dados) sem que esta esteja totalmente actualizada com a última revisão. A revisão de uma directoria de trabalho define um conjunto de entradas e de propriedades, e portanto, fazer `commit` a uma alteração às propriedades de uma directoria desactualizada (*out-of-date*) pode destruir propriedades que o utilizador ainda não saiba que existem.

A.8.6 Revisões: Números, Palavras-chave e Datas

Antes de se apresentar os exemplos de uso do SVN, convém reter mais algumas noções sobre os números de revisão, nomeadamente como se identificam as revisões num repositório.

As revisões são especificadas usando o *switch* `--revision` (ou `-r`) seguido do número de revisão que se pretende (`-r NUM-REV`), ou especificando um intervalo separando duas revisões por dois pontos (`-r NUM-REV1:NUM-REV2`). Posteriormente, o Subversion identifica estas revisões pelo número, pela data ou por uma palavra-chave.

A.8.7 Números de Revisão

Quando se cria um repositório SVN, este começa a sua vida útil com a revisão zero e cada *commit* sucessivo aumenta o número de revisão numa unidade. Após o *commit* ter terminado, o cliente de SVN informa o novo número de revisão:

```
$ svn commit --message "Corrigi o bug 123!!"
Sending ficheiroA.c
Transmitting file data .........

Committed revision 12.
```

Se a qualquer momento no futuro se quiser referenciar esta correcção, pode-se referenciá-la como a revisão 12.

O cliente de SVN percebe um número de palavras-chave relacionadas com as revisões. Estas palavras-chave podem ser usadas em vez dos números de revisão com o *switch* *--revision*, e são depois resolvidas de novo para números específicos de revisão pelo Subversion, sendo elas as seguintes:

- *HEAD*: a última (mais recente) revisão no repositório;
- *BASE*: o número de revisão de um item na cópia de trabalho. Se o item foi modificado localmente, a versão *BASE* refere-se ao item sem essas modificação locais;
- *COMMITTED*: a mais recente revisão antes, ou igual a *BASE*, de um item ter sido alterado;
- *PREV*: a revisão imediatamente antes da última revisão na qual um item foi alterado (tecnicamente é igual a *COMMITTED* - 1).

Eis alguns exemplos do uso de palavras-chave de revisão:

```
$ svn diff --revision PREV:COMMITED ficheiroA.c
```

Mostra a última alteração feita ao *ficheiroA.c* (último *commit*).

```
$ svn log --revision HEAD
```

Mostra o *log* do último *commit* feito no repositório.

```
$ svn diff --revision HEAD ficheiroA.c
```

Compara o ficheiro local *ficheiroA.c* (com eventuais alterações locais) com a última versão no repositório.

```
$ svn diff --revision BASE:HEAD ficheiroA.c
```

Compara a versão do *ficheiroA.c* (sem alterações locais feitas pelo utilizador) com a última versão no repositório.

```
$ svn log --revision BASE:HEAD
```

Mostra todos os logs dos *commit*s feitos desde a última vez que o utilizador actualizou a cópia de trabalho local.

```
$ svn update --revision PREV ficheiroA.c
```

Retrocede a última alteração feita ao *ficheiroA.c* (a revisão do *ficheiroA.c* é decrementada).

Estas palavras-chave permitem realizar várias operações importantes e rapidamente sem estar a ver especificamente quais os números de revisão que interessam da cópia de trabalho ou ainda qual é o último número de revisão da cópia de trabalho local.

A.8.8 Datas de Revisão

Quando se especifica um número de revisão ou uma palavra-chave de revisão, pode-se também especificar uma data dentro de { }. Pode-se ainda aceder a um intervalo de alterações no repositório usando em conjunto datas e números de revisão.

Eis alguns exemplos dos formatos de datas que o Subversion aceita. Não esquecer de usar { } a rodear as datas.

```
$ svn checkout --revision {2006-10-25}
$ svn checkout --revision {15:30}
$ svn checkout --revision {15:30:00.200000}
$ svn checkout --revision {"2006-10-25 15:30"}
$ svn checkout --revision {"2006-10-25 15:30 +0230"}
$ svn checkout --revision {2006-10-25T15:30}
$ svn checkout --revision {2006-10-25T15:30Z}
$ svn checkout --revision {2006-10-25T15:30-04:00}
$ svn checkout --revision {20061025T1530}
$ svn checkout --revision {20061025T1530Z}
$ svn checkout --revision {20061025T1530-0500}
```

Quando se especifica uma data de revisão, o Subversion encontra a versão mais recente do repositório naquela data:

```
$ svn log --revision {2006-10-25}
------------------------------------------------------------------------
r12 | ira | 2006-10-25 15:30:00-0600 (Wed, 25 Oct 2006) | 14 lines
...
```

Se se especificar uma única data sem incluir uma hora do dia (por exemplo, 2006-10-25), pode pensar-se que o Subversion deverá dar a última revisão que aconteceu no dia 25 de Outubro. Em vez disso, obtém-se uma revisão de dia 24 ou ainda anterior a este dia. Recorde-se que o Subversion encontra a revisão mais recente do repositório até à data que se introduziu. Quando se deu a data de 2006-10-25, o Subversion assumiu a hora de 00:00:00, por isso ao pesquisar pela revisão mais recente, não devolveu nada no dia 25. Se se quiser incluir o dia 25 na pesquisa, pode-se especificar o dia 25 com o tempo ({2006-10-25 23:59}), ou apenas ({2006-10-26}).

Pode-se também usar um intervalo de datas. O Subversion encontra todas as revisões entre as duas datas inclusivé:

```
$ svn log --revision {2006-10-14}:{2006-10-25}
```

Podemos também misturar datas e números de revisão, como mencionado em cima:

```
$ svn log --revision {2006-10-14}:1453
```

Nota: o *timestamp* de uma revisão é guardado como uma propriedade da revisão – é uma propriedade sem versão, que pode ser modificada. Uma vez que podem ser alterados os *timestamps* (representando assim um momento cronológico que não é correcto), ou até mesmo removidos, o Subversion não conseguirá converter correctamente datas para números de revisão se estas propriedades forem modificadas.

A.8.9 Estado das cópias de trabalho

Para cada ficheiro numa directoria de trabalho, o Subversion grava duas informações na pasta administrativa, `.svn/`:

- A revisão em que se está a trabalhar num determinado ficheiro (a chamada revisão de trabalho desse ficheiro), e
- Um *timestamp* que guarda a data da última actualização feita ao repositório para esse ficheiro local;

Dadas estas duas informações, e ao comunicar com o repositório, o Subversion pode indicar em que estado um ficheiro local se encontra (de um total de 4 estados possíveis):

- **Sem modificações, e actualizado**: o ficheiro não tem modificações na directoria de trabalho, e não houve `commit`s feitos ao repositório desde a revisão de trabalho desse ficheiro. Um `svn commit` a este ficheiro não vai alterar nada no seu conteúdo e um `svn update` também não vai alterar nada à revisão do ficheiro;
- **Com modificações locais, e actualizado**: o ficheiro foi alterado na directoria de trabalho, mas não foi feito nenhum `commit` ao repositório desde a sua revisão anterior. Há alterações locais às quais não foi feito nenhum `commit`, logo um `svn commit` vai colocar as alterações no repositório de SVN, e um `svn update` não vai alterar nada ao conteúdo desse ficheiro;
- **Sem modificações locais, e desactualizado**: o ficheiro não teve alterações na directoria de trabalho, mas foi modificado por outro utilizador no repositório. O ficheiro deverá eventualmente ser actualizado, para o colocar com a última revisão de acordo com a revisão no repositório. Um `svn commit` ao ficheiro não fará nada, mas um `svn update` vai colocar esse ficheiro com as últimas modificações feitas na cópia de trabalho;
- **Com modificações locais, e desactualizado**: O ficheiro foi alterado pelo utilizador na cópia de trabalho local, e no repositório por outro utilizador. Um `svn commit` do ficheiro vai falhar com o erro de desactualizado[1]. O ficheiro deverá ser actualizado primeiro; um `svn update` vai tentar fundir ou unificar as alterações públicas do repositório com as alterações locais. Se o Subversion não conseguir fazer essa fusão de uma forma automática, então acontecerá um conflito e caberá ao utilizador resolver manualmente este conflito.

O comando `svn status` mostra o estado de um item numa cópia de trabalho e é bastante útil para ver o estado dos ficheiros numa directoria de trabalho local.

A.8.10 Cópias de trabalho com revisões diferentes

O Subversion tenta sempre manter uma certa flexibilidade, tanto quanto possível, para permitir que se possa trabalhar numa cópia de trabalho com ficheiros e directorias com diferentes números de revisão. Infelizmente, isto pode ser um pouco confuso, mas facilmente se percebe a utilidade de ter números de revisão diferentes numa cópia de trabalho. Uma das regras fundamentais do Subversion é que um *push* ao repositório não causa um *pull*, e vice-versa. Ou seja, quando se submetem alterações ao repositório (*push*) isso não significa que se vá receber as alterações feitas pelos outros utilizadores (*pull*). E se um utilizador estiver a fazer modificações que ainda não tenham sido passadas ao repositório com um `commit`, fazendo `svn update` vai unificar as últimas alterações do repositório com as alterações em curso deste utilizador, sem que o utilizador seja obrigado a fazer `commit` às alterações que estava a fazer.

O efeito principal desta regra é que uma cópia de trabalho tem que ter um trabalho extra em tomar nota de todas as revisões diferentes que há na cópia de trabalho, e de ser flexível com as diferentes revisões para permitir que existam localmente. O que também complica neste processo é o facto de as próprias directorias terem também versões. Por exemplo, suponha que se tem uma cópia inteira de trabalho com a revisão 10. Edita-se o

[1] *out-of-date*

ficheiro `xpto.html` e faz-se e em seguida um `svn commit`, que cria a revisao 16 no repositório. Após o `commit` ter sido bem sucedido, muitos utilizadores pensarão que a cópia de trabalho local está toda com a revisão 16, o que não acontece! Qualquer número de alterações pode ter sido feito ao repositório entre as revisões 10 e 16. O cliente de SVN não sabe de nenhumas dessas alterações ao repositório, uma vez que o utilizador local ainda não fez nenhum `svn update`, e um `svn commit` não traz para a cópia de trabalho as alterações feitas aos outros ficheiros entretanto. Por outro lado, se um `svn commit` fizesse um *download* automático das novas alterações que estavam no repositório, então isso iria colocar a cópia de trabalho local na revisão 16 – mas aí estaríamos a quebrar a regra de que um *push* e um *pull* são tarefas separadas. Portanto, a única acção segura que o cliente de SVN tem que fazer é de colocar o ficheiro `xpto.html` com a revisão 16. O resto da cópia de trabalho fica com a revisão 10. Só depois de se fazer um `svn update` é que as restantes alterações vão ser descarregadas, e toda a cópia de trabalho passa a ficar marcada com a revisão 16.

A.9 Subversion em acção

Nesta secção vamos passar da teoria à prática, exemplificando a utilização do Subversion para que se consiga ter uma percepção mais abrangente das vantagens de utilização deste SCM. O seguinte exemplo assume que já esteja instalado o `svn` – o cliente de linha de comandos do Subversion – e o `svnadmin` – a ferramenta administrativa – e que estejam prontos a usar. Assume também que se esteja a usar o Subversion 1.5 ou mais recente (executar `svn --version` para verificar a versão).

A.9.1 Cópias de trabalho

Como foi indicado acima, uma cópia de trabalho para o Subversion é apenas uma directoria no sistema de ficheiros local de trabalho de cada utilizador, que contém uma colecção de ficheiros. Cada utilizador pode editar os ficheiros como quiser, e se forem ficheiros de código-fonte de uma qualquer linguagem de programação, o utilizador pode compilar o programa da maneira habitual com os editores ou *IDE*'s que preferir. A cópia de trabalho é uma área de trabalho privada: o Subversion nunca vai incorporar nesta área de trabalho as alterações das outras pessoas, ou colocar as alterações feitas nessa área de trabalho visíveis aos outros utilizadores, até que se indique explicitamente ao Subversion para o fazer. É possível ainda ter várias cópias de trabalho do mesmo projecto. Após terem sido feitas alterações aos ficheiros na cópia de trabalho pessoal de cada utilizador e de se verificar que estas funcionam correctamente, o Subversion fornece os comandos para se publicar as alterações para os outros utilizadores que trabalham no mesmo projecto (ao escrever para o repositório). Se outras pessoas publicarem as suas próprias alterações, o Subversion fornece comandos para unificar essas alterações na cópia de trabalho de cada um (ao ler do repositório).

Uma cópia de trabalho contém também alguns ficheiros extra, criados e mantidos pelo Subversion, para ajudar na execução dos comandos do Subversion. Em particular, cada directoria na cópia de trabalho contém uma subdirectoria chamada `.svn`, que é também conhecida por *directoria administrativa* da cópia de trabalho. Os ficheiros na directoria administrativa ajudam o Subversion a reconhecer que ficheiros contêm alterações por publicar, e que ficheiros estão desactualizados em relação ao trabalho feito pelos outros

utilizadores. Um repositório típico de Subversion contém frequentemente ficheiros (ou código-fonte) de vários projectos; normalmente, cada projecto é uma sub-directoria na árvore do sistema de ficheiros do repositório. Por isso, a cópia de trabalho de um utilizador corresponde geralmente a uma sub-árvore particular do repositório global.

A título de exemplo, suponhamos que temos um repositório Subversion que contém dois projectos de *software*, `ProjX` e `ProjY`. Cada projecto existe na sua própria sub-directoria. Para ter uma cópia de trabalho, um utilizador tem de fazer *checkout* a uma sub-árvore do repositório (o termo *checkout* pode parecer que se está a fazer algum bloqueio ou a reservar recursos, mas na verdade não; simplesmente cria uma cópia privada do projecto na máquina local do utilizador). Por exemplo, se o utilizador fizer *checkout* ao `ProjX/`, fica com uma cópia local deste projecto:

```
$ svn checkout http://svn.exemplo.com/repositorio/ProjX
A ProjX/pom.xml
A ProjX/ClasseA.java
A ProjX/LEIAME.txt
...
Checked out revision 29.

$ ls -A ProjX
pom.xml ClasseA.java LEIAME.txt ... .svn/
```

A lista que começa com `A` significa que o Subversion está a adicionar esses ficheiros à cópia de trabalho pessoal do utilizador. O utilizador possui agora uma cópia pessoal da directoria `ProjX/` do repositório, com uma entrada adicional – `.svn` – que contém informação extra necessária ao Subversion, como mencionado acima. Nunca se deverá apagar as pastas `.svn`; estas pastas ocultas são a porta de entrada da comunicação cliente-servidor de SVN.

Supondo que um utilizador fazia alterações no ficheiro `ClasseA.java`, a directoria `.svn` contém a data original de modificação deste ficheiro e do conteúdo que tinha nessa data, e o Subversion pode assim indicar que o ficheiro foi alterado. No entanto, o Subversion não torna as alterações do utilizador públicas até que o utilizador o indique. Publicar as alterações introduzidas em determinado ficheiro para ficarem visíveis aos outros utilizadores é feito através do comando *commit* ao repositório:

```
$ svn commit ClasseA.java

Sending ClasseA.java
Transmiting file data .........

Committed revision 30.
```

Agora que as alterações à `ClasseA.java` foram colocadas no repositório, qualquer utilizador que faça *checkout* à cópia de trabalho do `ProjX/` vai poder ver as alterações introduzidas na última versão do ficheiro `ClasseA.java`. Suponhamos agora que a Eunice fez *checkout* à sua cópia de trabalho do `ProjX/` ao mesmo tempo que o Miguel.

Quando o Miguel faz *commit* às alterações do ficheiro `ClasseA.java`, a cópia de trabalho da Eunice fica inalterada; o Subversion apenas modifica uma cópia de trabalho a pedido do utilizador. Para colocar o seu projecto actualizado, a Eunice faz um pedido ao Subversion para fazer *update* à sua cópia de trabalho, usando para isso o comando *update*. Isto vai incorporar as alterações do Miguel no ficheiro `ClasseA.java`, bem como outras alterações feitas pelos outros utilizadores do projecto desde que a Eunice fez *checkout* ao repositório de Subversion.

```
$ pwd
/repositorio/ProjX/
$ ls -A
.svn/ pom.xml ClasseA.java LEIAME.txt ...
$ svn update
U ClasseA.java
Updated to revision 30.
```

A saída do comando `svn update` indica que o Subversion fez uma actualização aos conteúdos do ficheiro `ClasseA.java`. Note-se que a Eunice não precisou de especificar a que ficheiros fazer o update; o Subversion usa a informação da directoria `.svn`, e outras informações contidas no repositório, para decidir que ficheiros precisam de ser actualizados na cópia de trabalho da Eunice.

A.9.2 Criação de um repositório

O Subversion guarda todas versões do *software* num repositório central. Para começar o nosso exemplo, cria-se um novo repositório com o comando seguinte:

```
$ svnadmin create /opt/repositorio/
$ ls /opt/repositorio/
conf/ dav/ db/ format hooks/ locks/ README.txt
```

Este comando cria uma nova directoria (no exemplo acima, em `/opt/repositorio`) que contém um repositório de Subversion. Esta directoria contém para além de outros ficheiros, uma colecção de ficheiros de base de dados. Porém, não se acede aos ficheiros do *software* e às diferentes versões se explorarmos o repositório de Subversion acabado de criar. O Subversion não tem o conceito de *projecto*. O repositório é apenas um sistema de ficheiros virtual com versões, uma árvore de ficheiros que pode conter qualquer tipo de ficheiro que se queira. Alguns programadores preferem guardar apenas um só projecto num repositório, enquanto que outros guardam vários projectos no repositório guardando cada um deles em directorias separadas. De qualquer forma, o repositório apenas guarda ficheiros e directorias, cabe aos programadores saber que directorias e ficheiros pertencem a determinado projecto. Assim, tudo o que realmente se está a falar quando se fala de repositórios de Subversion é de uma colecção de directorias e de ficheiros à qual o Subversion não conhece a que projecto pertence cada ficheiro ou directoria.

Supondo que o *software* a partir do qual se quer começar a depender do Subversion está numa pasta designada `/opt/softwareXPTO/`, começa-se por criar três directorias

com os nomes `branches`, `tags` e `trunk`. A pasta `trunk` deverá conter todo o código de *software* e ficheiros dependentes, enquanto que as pastas `branches` e `tags` deverão estar vazias:

```
/opt/softwareXPTO/branches/
/opt/softwareXPTO/tags/
/opt/softwareXPTO/trunk/
                ficheiroA.c
                ficheiroB.h
                ficheiroC.html
                Makefile
                configure
                ...
```

Uma vez criadas estas sub-directorias, importa-se a pasta do *software* para o repositório criado com o comando `svn import`:

```
$ svn import /opt/softwareXPTO/ \
        file:///opt/repositorio/softwareXPTO \
        -m "import inicial para o repositório"

Adding /opt/softwareXPTO/branches
Adding /opt/softwareXPTO/tags
Adding /opt/softwareXPTO/trunk
Adding /opt/softwareXPTO/trunk/ficheiroA.c
Adding /opt/softwareXPTO/trunk/ficheiroB.h
Adding /opt/softwareXPTO/trunk/ficheiroC.html
Adding /opt/softwareXPTO/trunk/Makefile
Adding /opt/softwareXPTO/trunk/configure
...

Committed revision1.
```

O repositório contém nesta altura a árvore de ficheiros pretendida. Como foi mencionado acima, não se acedem aos ficheiros do projecto de *software* olhando directamente para o conteúdo do repositório; o código fica armazenado numa base de dados dentro do repositório. Para começar a manipular os dados no repositório, é necessário criar uma cópia de trabalho (*working copy*) dos dados, como se fosse uma área de trabalho privada. Para tal, pede-se ao Subversion para fazer um *checkout* da cópia de trabalho da directoria `/opt/repositorio/softwareXPTO/trunk` no repositório:

```
$ svn checkout file:///opt/repositorio/softwareXPTO/trunk \
        softwareXPTO

A softwareXPTO/ficheiroA.c
A softwareXPTO/ficheiroB.h
A softwareXPTO/ficheiroC.html
A softwareXPTO/Makefile
```

```
A softwareXPTO/configure
...

Checked out revision 1.
```

Após esta operação, temos uma cópia pessoal de parte do repositório numa directoria chamada `softwareXPTO`. Pode-se editar os ficheiros na cópia de trabalho e depois fazer *commit* a essas alterações de novo para o repositório.

- Dentro da pasta da cópia de trabalho editar e alterar o conteúdo de um ficheiro;
- Fazer `svn diff` para ver uma saída das diferenças unificadas das alterações feitas;
- Fazer `svn commit` para introduzir a nova versão dos ficheiros no repositório;
- Fazer `svn update` para colocar na cópia de trabalho a versão mais actual do repositório.

A partir deste ponto pode tornar-se o repositório acessível a outros utilizadores e/ou programadores numa rede de computadores, fazendo uso de um servidor HTTP, por exemplo o servidor HTTP da Apache.

A.9.3 Ciclo Normal de Trabalho

Na maior parte das vezes, começa-se a usar um repositório de Subversion fazendo um checkout ao projecto em questão. Ao fazer isto, é criada na máquina local uma cópia do projecto. Esta cópia contém a HEAD (última revisão) do repositório SVN que é especificado na linha de comandos:

```
$ svn checkout http://svn.exemplo.pt/repositorio/trunk

A trunk/subversion.dsw
A trunk/svn_check.dsp
A trunk/Makefile
A trunk/configure
A trunk/ficheiroA.c
...
Checked out revision 327.
```

Apesar do exemplo acima fazer checkout da directoria `/trunk`, pode-se igualmente escolher uma sub-directoria do repositório ao especificar o subdirectório no URL:

```
$ svn checkout http://svn.exemplo.pt/repositorio/trunk \
        /documentacao/pdfs

A trunk/documentacao/pdfs/faq.pdf
A trunk/documentacao/pdfs/userguide.pdf
A trunk/documentacao/pdfs/manual-de-instalacao.pdf
```

```
...
Checked out revision 327.
```

Uma vez que o Subversion usa o modelo *copia-modifica-unifica* em vez do modelo *bloqueia-modifica-desbloqueia*, o utilizador pode logo começar a fazer alterações aos ficheiros e directorias na sua cópia de trabalho. Esta cópia de trabalho é igual a qualquer outra colecção de ficheiros e directorias no computador. O utilizador pode editar e alterar os ficheiros, navegar e mover pastas, ou até mesmo apagar a cópia de trabalho inteira e trabalhar numa nova. Apesar de realmente ser uma colecção de ficheiros e directorias como qualquer outra, é preciso sempre avisar o Subversion se se quiser rearranjar qualquer ficheiro ou directoria dentro da cópia de trabalho. Por isso, deve-se sempre usar os comandos `svn copy` ou `svn move` em vez dos comandos normais (`Ctrl-C` ou `Ctrl-X`, por exemplo) para que essas alterações sejam reflectidas a todos os utilizadores quando fizerem um `svn update` nas suas cópias de trabalho.

Cada directoria na cópia de trabalho local contém uma área administrativa, uma sub-directoria chamada `.svn`. Normalmente, um comando de listagem de directórios não mostra esta sub-directoria, mas é no entanto uma directoria muito importante que não se deve nunca apagar ou alterar. O Subversion depende desta directoria para gerir a cópia de trabalho de cada utilizador.

Pode-se também fazer um *checkout* especificando uma nova directoria que vai passar a ser a nova cópia de trabalho depois do URL do repositório, como se indica de seguida:

```
$ svn checkout \
        http://svn.exemplo.pt/repositorio/trunk/documentacao/pdfs \
        directoriaNova

A directoriaNova/faq.pdf
A directoriaNova/userguide.pdf
A directoriaNova/manual-de-instalacao.pdf
...
Checked out revision 327.
```

Isto coloca a cópia de trabalho numa directoria chamada `directoriaNova` em vez da directoria trunk como tínhamos visto previamente.

O Subversion tem bastantes funcionalidades, opções, possibilidades de uso, mas para uma utilização diária só se irá usar um sub-conjunto limitado das funcionalidades do Subversion.

O ciclo de trabalho é tipicamente o seguinte:

- **Fazer um update à cópia de trabalho**
  - `svn update`
- **Fazer as alterações necessárias aos ficheiros da cópia de trabalho**
  - `svn add`
  - `svn delete`

  - `svn copy`
  - `svn move`

- **Examinar as alterações feitas**
  - `svn status`
  - `svn diff`
  - `svn revert`

- **Unificar as alterações feitas por outros utilizadores na cópia de trabalho local**
  - `svn update`
  - `svn resolved`

- E por fim, e só nesta altura, **fazer uma submissão das alterações feitas na cópia de trabalho local para o repositório de SVN**
  - `svn commit`

A Figura A.5 exemplifica melhor o ciclo normal de desenvolvimento com o Subversion.

A.10 Propriedades do Subversion

O Subversion permite criar propriedades com versão nos ficheiros e directorias (com os nomes que se quiser), bem como propriedades sem versão nas revisões. A única restrição encontra-se nas propriedades com o prefixo `svn:` que são propriedades reservadas ao Subversion nesse espaço de nomes (*namespace*). Enquanto que as propriedades fora deste espaço de nomes podem ser usadas para controlar o comportamento do Subversion, os utilizadores não podem criar ou alterar propriedades do espaço de nomes `svn:`. Estas são:

- **Propriedades com versão**
  - `svn:executable` - Se está presente num ficheiro, o cliente de SVN fará o ficheiro executável em cópias de trabalho em ambiente Unix; Em ambiente Windows, para ser executável basta ter a extensão `.exe` ou `.bat`.
  - `svn:mime-type` - Se está presente num ficheiro, o valor indica o tipo MIME [Wik09k] do ficheiro. Isto permite ao cliente de SVN decidir se uma fusão (*merging*) contextual é possível de ser feita durante um *update*, e pode também afectar como o ficheiro se comporta quando é procurado via *web browser*;
  - `svn:ignore` - Se está presente numa directoria, o valor é uma lista de padrões de ficheiros sem versão a serem ignorados pelo subcomando `svn status` ou outros; Por exemplo, ”*.log *.dmp core.*”;
  - `svn:keywords` - Se está presente num ficheiro, o valor diz ao cliente de SVN como expandir palavras-chave (*keywords*) particulares dentro do ficheiro;

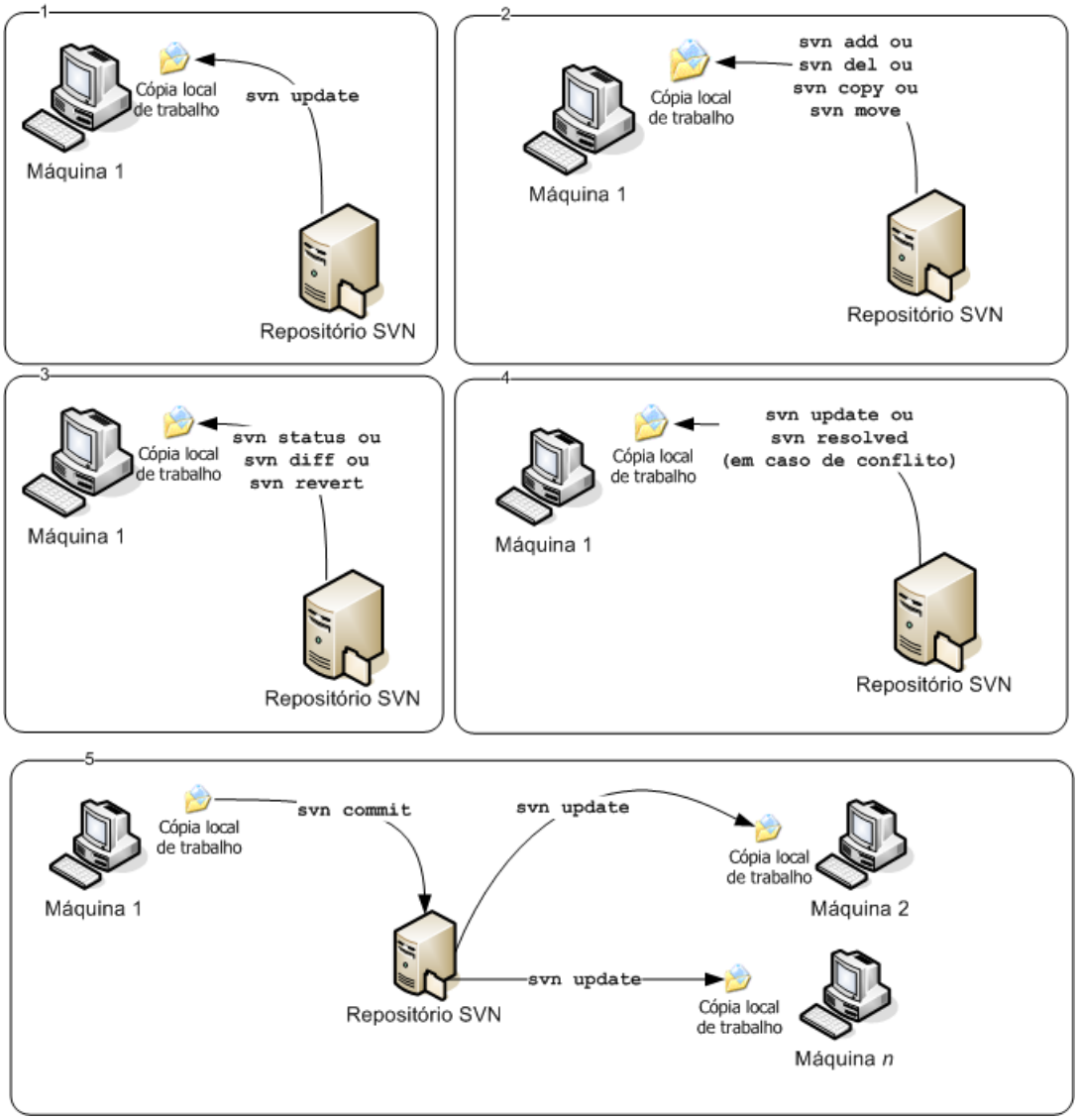

Figura A.5: Ciclo normal de trabalho com o SVN.

- `svn:eol-style` - Se está presente num ficheiro, o valor diz ao cliente de SVN como manipular o fim de uma linha no ficheiro na cópia de trabalho; Por exemplo, `CRLF` (em Windows) ou `CR` (em Unix/Linux) [Wik09l];
- `svn:externals` - Se está presente numa directoria, o valor é uma lista com vários caminhos (*paths*) e URLs a que o cliente de SVN deverá fazer *checkout*; Esta propriedade é muito útil quando se precisa de usar código-fonte de terceiras fontes mas que não precisamos que estejam versionadas no repositório de SVN;
- `svn:special` - Se está presente num ficheiro, indica que o ficheiro não é um ficheiro habitual, mas sim um link simbólico ou outro tipo especial de objecto;
- `svn:needs-lock` - Se está presente num ficheiro, indica ao cliente de SVN para colocar o ficheiro só para leitura na cópia de trabalho, como uma nota de que esse ficheiro deverá ser bloqueado (*locked*) antes da sua edição (para

desta forma ganhar propriedades de escrita e garantir que só um utilizador pode alterar esse ficheiro).

- **Propriedades sem versão**
  - `svn:author` - Se está presente, contém o nome do utilizador autenticado que criou a revisão (se não estiver presente, a revisão foi feita anonimamente);
  - `svn:date` - contém o tempo UTC [Wik09c] em que a revisão foi criada, no formato ISO [Wik09g]. O valor vem do relógio do servidor;
  - `svn:log` - contém a mensagem de *log* que descreve a revisão; Esta mensagem pode ser alterada após um *commit*, caso surja algum engano ou erro na mensagem que se queria ter colocado no *commit*;
  - `svn:autoversioned` - Se está presente, a revisão foi criada com a funcionalidade de versionamento automático.

A.10.1 URLs dos repositórios Subversion

Os repositórios de Subversion podem ser acedidos através de vários métodos – localmente no disco (directamente) ou através de vários protocolos de rede (indirectamente). A localização de um repositório, no entanto, é sempre um URL [Wik09q]. Em seguida apresenta-se uma tabela que descreve como os diferentes URLs mapeam os diferentes métodos de acesso a um repositório Subversion.

Esquema	Método de Acesso
`file://`	Acesso directo ao repositório (disco local)
`http://`	Acesso via protocolo `WebDAV` [Wik09s] para o servidor HTTP
`https://`	O mesmo que para `http://`, mas com encriptação `SSL` [Wik09p]
`svn://`	Acesso via protocolo `svn://` a um servidor de SVN a correr `svnserve`
`svn+ssh://`	O mesmo que `svn://`, mas através de um túnel `SSH` [Wik09o]

Tabela A.1: Protocolos de acesso a um repositório de Subversion.

Se se usar o SVN em Windows em ambiente DOS [Wik09d], as barras devem ser como numa directoria em Linux (/) e não da forma nativa (\). Por exemplo, `file:///repositorio/pasta/` e não `file:///repositorio\pasta\`.

A.11 Ajuda do SVN

Um dos mais importantes comandos do SVN é o svn help. A qualquer momento, um

```
$ svn help sub-comando
```

descreve a sintaxe, os *switches* que podem ser usados e o comportamento normal desse sub-comando.

A.12 Conclusões

O controlo de versões de *software* é imprescindível no desenvolvimento de *software*. Não só permite agilizar o mesmo como também fornece segurança aos programadores para avançar com as suas implementações sem risco de se perder código. O controlo de todo o *software*, desde a ideia inicial até às sucessivas iterações por que passa uma ideia, é registado pelo VCS. Se houver necessidade de voltar atrás, é rápido, simples e é garantido que se tem a versão anterior pretendida sem perda de código.

O Subversion na sua globalidade é um excelente VCS, que tem visto um aumento de popularidade entre programadores de todo o mundo. Para além de fornecer características importantes para a produtividade do *software*, tais como a atomicidade dos commits, a gestão das fusões de código de diferentes programadores, o versionamento de directorias e o custo das operações ser proporcional ao tamanho das alterações (e não ao tamanho dos ficheiros), é ainda *software* livre que pode ser usado com outras ferramentas (Apache HTTP [Fou09a], Trac [Sof09c]) para se ter uma solução valiosa face a sistemas comerciais (sobre este aspecto, ver Testemunhos).

Para além disto, o Subversion é desenvolvido por milhares de programadores em todo o mundo, que trabalham para o aumento da qualidade deste VCS, e é suportado por uma extensa comunidade online. Uma forma rápida de ficar a saber trabalhar com o Subversion é através dos exemplos do livro online *SVN Red Book* [BCS08] e das questões levantadas nas *mailing lists* do Subversion [Col09d].

Como limitações, o Subversion tem apenas controlo de acessos dos utilizadores por directoria e não tem ainda uma maior "granularidade", no que toca ao acesso dos ficheiros. Um exemplo disto é quando existe um projecto com pastas lib, include, bin, que deviam de ter segurança implícita (ou seja, o nome das pastas já diz implicitamente o que certos actores podem fazer, como os programadores só acederem à pasta `lib` e `include`, e os *testers* do projecto acederem exclusivamente à pasta `bin`, onde se encontram os binários a serem alvo de testes).

Outro problema actual com o Subversion é a administração de um repositório, em que por exemplo, não é de todo fácil apagar revisões que são lixo. Se uma revisão não interessar ou representar um retrocesso para outra que já existia, não é fácil apagar essa revisão do repositório. Isto é um pouco deselegante, uma vez que o repositório fica com revisões que não tem interesse de manter.

Por fim, outro dos problemas prende-se com as pastas ocultas `.svn`, pois se o programador acidentalmente apagar esta pasta oculta, perde a forma de indicar ao servidor de SVN que fez modificações nessa pasta. Existe um procedimento para reverter esta situação, mas não é simples o suficiente e poderia estar já integrado à partida no Subversion.

Independentemente destas desvantagens, o Subversion é um VCS que representa uma *mais-valia*, não só para o ambiente empresarial, mas também para o dia-a-dia de um programador ou ainda aluno de uma faculdade (uma excelente ideia seria a de cada aluno ter uma conta SVN e fazer *commits* dos trabalhos para um repositório central SVN da faculdade). De notar que o Subversion pode ser usado para qualquer projecto e não tem necessariamente de ser um projecto de *software*; o importante é a garantia que o Subversion fornece em guardar as alterações que se façam num projecto e o conjunto rápido de informação que se reúne sobre essas mesmas alterações. A partir daí, o projecto em si pode ser de qualquer área de negócio.

Unless in communicating with it [a computer] one says exactly what one means, trouble is bound to result.

A. Turing

B Instalação do Subversion

A instalação do SVN é rápida e fácil, tanto em ambiente Linux como em Windows. Este anexo serve como guia rápido de instalação do SVN nos dois sistemas operativos: Linux e Windows. É também descrita a forma de instalar e configurar o servidor HTTP da Apache para aceitar URLs provenientes de repositórios SVN e também como instalar o Trac [Sof09c], um sistema de *bug-tracking* que serve como interface web para o Subversion. Por fim, indica-se como instalar um dos plugins existentes de SVN para o IDE Eclipse [Com09], de nome Subclipse [Col09a].

B.1 Instalação do SVN em Linux

A instalação do Subversion em Linux (para a distribuição Ubuntu [Can09]) é extremamente simples e funcional. Basta usar o comando seguinte:

```
$ sudo apt-get install subversion libapache2-svn
```

B.2 Instalação do SVN em Windows

A instalação em Windows pode ser feita de duas maneiras: instalando os binários do cliente/servidor de SVN (e passa-se a usar apenas o subversion numa *shell* DOS, o que não é muito interessante para os utilizadores do Windows), ou instala-se um cliente gráfico para Windows: um dos melhores actualmente é o TortoiseSVN. Vejamos a seguir como instalar o TortoiseSVN [Col09e] (disponível apenas para Windows):

1. Descarregar o TortoiseSVN no site oficial de downloads [Col09f];
2. Duplo-clique no instalador .msi e resta apenas seguir as instruções apresentadas;
3. Um *reboot* será necessário para o TortoiseSVN ser correctamente instalado.

Uma das principais vantagens de uso do TortoiseSVN é a integração que tem no *Windows explorer* (um clique no botão direito do rato faz surgir no menu de contexto as opções do TortoiseSVN) independentemente de outros clientes gráficos que estejamos a usar. Outra grande vantagem é a sobreposição de ícones no Windows: nas pastas que sejam de cópias locais de trabalho SVN, os ícones das pastas e dos ficheiros são alterados para indicar visualmente o estado da cópia local de trabalho; em suma, o programador vê instantaneamente o estado da sua cópia local de trabalho sem necessitar de fazer um `svn status`, o que é realmente muito útil (Figura B.1).

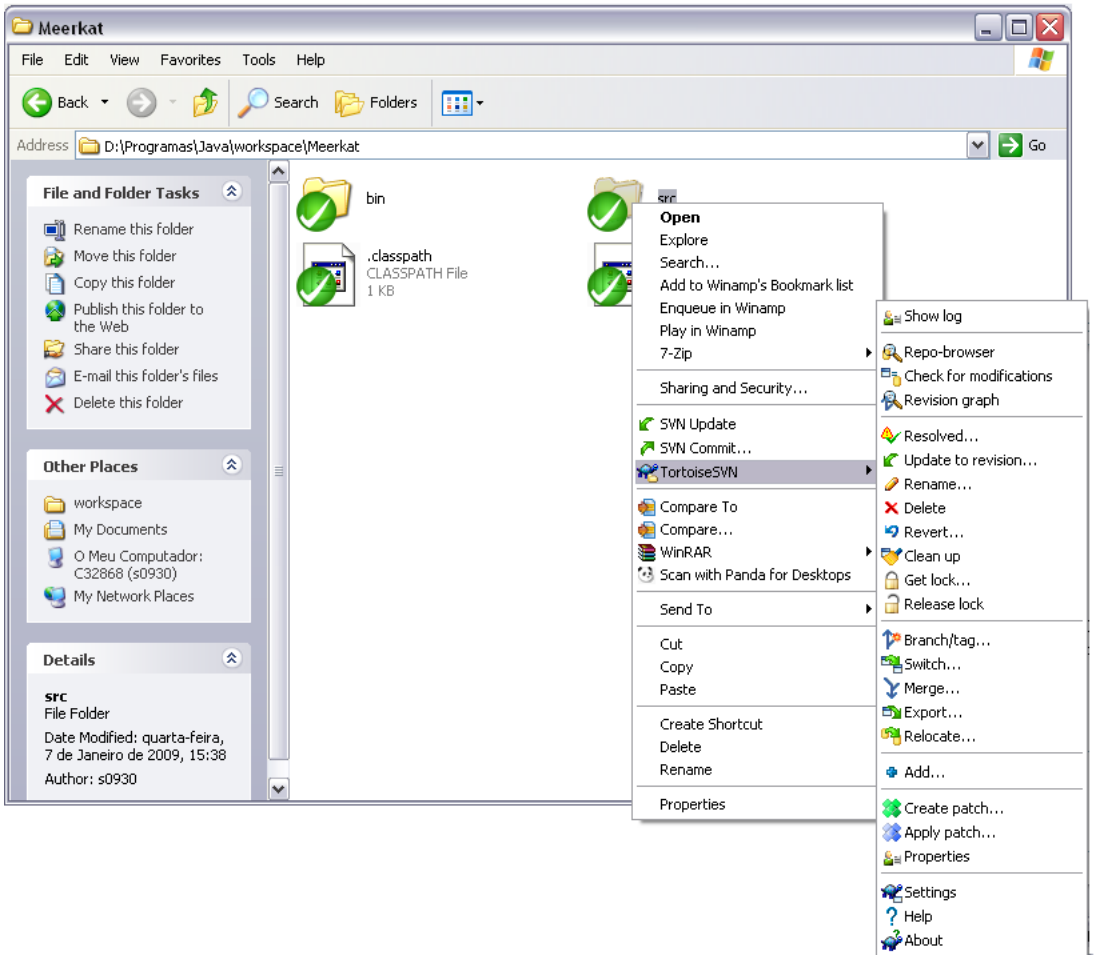

Figura B.1: Opções de SVN através do menu de contexto do TortoiseSVN. Note-se a alteração nos ícones das pastas e ficheiros no *Windows Explorer*.

B.2.1 Componentes do Subversion

Uma vez instalado o Subversion, temos uma série de diferentes componentes deste *software* ao nosso dispôr:

- `svn`: o programa cliente de linha de comandos;
- `svnversion`: o programa para reportar o estado (as revisões) de uma cópia de trabalho;

- `svnlook`: a ferramenta para inspeccionar o estado de um repositório de Subversion;
- `svnadmin`: a ferramenta para criar, alterar ou reparar um repositório de Subversion;
- `svndumpfilter`: o programa para filtrar os *dump streams* do Subversion;
- `mod_dav_svn`: um módulo de plugins para o servidor HTTP da Apache, usado para colocar o repositório disponível a outros utilizadores através de uma rede;
- `svnserve`: um programa servidor, que corre como um processo (*daemon*) ou pode ser também invocado via SSH [Wik09o]; é outra forma de tornar o repositório disponível a outros utilizadores através de uma rede, através do protocolo `svn://`, em vez de se usar o servidor HTTP da Apache.

B.2.2 Configuração do servidor de HTTP da Apache

Nesta secção é detalhado o modo de configurar o servidor HTTP da Apache para usar URLs que pertençam a repositórios SVN, usufruindo deste modo da autenticação e autorização fornecidos pelo servidor HTTP. Os passos que se seguem são para Linux (distribuição Ubuntu), mas para Windows é em tudo semelhante (os comandos para criar e popular um repositório podem ser todos feitos recorrendo ao TortoiseSVN).

Vamos começar por criar um repositório:

```
$ svnadmin create /home/projecto/ProjX
$ ls /home/projecto/ProjX
conf/  dav/  db/  format  hooks/  locks/  README.txt
```

Neste momento temos um repositório SVN para o ProjX, mas que está vazio, sem ficheiros. Para popular o repositório com ficheiros, teremos que ter outra pasta que tem todos os ficheiros que queremos adicionar ao projecto SVN:

```
$ ls -F /tmp/ProjX
trunk/ tags/ branches/
$ ls -F trunk
pasta1/ pasta2/ README.txt CHANGELOG ficheiro1.c ficheiro1.h
```

Esta pasta `/tmp/ProjX` tem todo o código que vamos querer importar para o repositório. Depois disso, podemos apagar esta pasta, pois todo o seu conteúdo já estará no repositório SVN. Resta portanto, importar o conteúdo desta pasta para o nosso repositório SVN:

```
$ svn import /tmp/ProjX file:///home/projecto/ProjX \
        -m "import inicial"
```

Não colocar / no fim dos caminhos (é /tmp/ProjX e não /tmp/ProjX/; o mesmo para o segundo caminho indicado); O nome para o projecto, neste caso ProjX, deverá ser coincidente nos dois caminhos apresentados.

Depois de importar os dados com sucesso, vamos transferir o conteúdo de ProjX para uma cópia local de trabalho:

```
$ svn checkout file:///home/projecto/ProjX/trunk nome-do-projecto


A  nome-do-projecto/pasta1
A  nome-do-projecto/pasta2
A  nome-do-projecto/README.txt
...
Checked out revision 1.
```

No checkout, o nome para a pasta local que vai conter o ProjX é o que o utilizador pretender, não tem de ser igual a ProjX.

B.2.3 Configuração do servidor HTTP da Apache

Resta apenas configurar o servidor HTTP da Apache. Para isso deve-se aceder ao ficheiro de configuração, apache2.conf (Em versões anteriores à 2.2 do servidor HTTP da Apache, o ficheiro apache2.conf é substituído pelo ficheiro httpd.conf), que fica em /etc/apache2 e adicionar as entradas:

```
LoadModule dav_module         modules/mod_dav.so
LoadModule dav_svn_module     modules/mod_dav_svn.so

...(na secção inicial de carregamento dos módulos)

<Location /ProjX>
  DAV svn
  SVNPath /home/projecto/ProjX
</Location>

...(na secção das Locations; Neste caso, URLs do tipo
http://localhost/ProjX vão ser tratados pelo Apache para
carregar com o módulo dav_svn o conteúdo do projecto ProjX.)
```

Reiniciamos o apache:

```
sudo /etc/init.d/apache2 restart
```

E temos neste momento o servidor Apache configurado para usar o repositório SVN ProjX! Basta abrir um browser e apontar para o URL http://localhost/ProjX. Mas, neste exemplo, não estamos a usufruir da autenticação. Para tal, alteramos de novo o ficheiro apache2.conf e adicionamos:

```
<Location /ProjX>
  DAV svn
  SVNPath /home/projecto/ProjX
  AuthType Basic
  AuthName "Repositorio para o ProjX"
  AuthUserFile /etc/svn-auth-file
</Location>
```

Criamos o ficheiro `/etc/svn-auth-file` e adicionamos credenciais para os utilizadores que pretendemos que tenham acesso ao servidor SVN (A opção `-c` é usada para criar o ficheiro se ainda não existir, e a opção `-m` é para usar encriptação MD5 [Wik09i] nas passwords.):

```
$ htpasswd -cm /etc/svn-auth-file utilizador1
New password: *****
Re-type new password: *****
Adding password for user utilizador1
$ htpasswd -m /etc/svn-auth-file utilizador2
New password: *****
Re-type new password: *****
Adding password for user utilizador2
```

Podemos ainda usufruir da autorização baseada em caminhos, em que dizemos ao servidor HTTP que determinado utilizador só tem acesso a determinada pasta. Para tal adicionamos ao ficheiro `apache2.conf`:

```
LoadModule authz_svn_module   modules/mod_authz_svn.so
...(na secção de carregamento dos módulos)

<Location /ProjX>
  DAV svn
  SVNPath /home/projecto/ProjX

  Require valid-user

  AuthType Basic
  AuthName "Repositorio para o ProjX"
  AuthUserFile /etc/svn-auth-file
  AuthzSVNAccessFile /etc/access-file
</Location>
```

E criamos o ficheiro `/etc/access-file` indicando que acesso por pasta cada utilizador tem:

```
[ProjX:/trunk/pasta1]
utilizador1 = rw
utilizador2 = r
```

Neste exemplo, indica-se que o utilizador1 tem permissões de escrita e de leitura à pasta1 mas o utilizador2 só pode ler os conteúdos desta pasta.

B.3 SVN no trabalho colaborativo (Trac)

O Trac [Sof09c] é um sistema de gestão de projectos e de *bug-tracking* que integra numa wiki o conteúdo de repositórios SVN para uma rápida visualização do que se passa nos mesmos. Para além de possibilitar aceder a todo o código de um repositório SVN (com *highlight* de código), permite introduzir na wiki os objectivos e os prazos para o projecto, os *bugs* encontrados, aferir o que cada utilizador tem alterado no projecto, entre outras funcionalidades muito interessantes para uma visualização intuitiva e rápida dos projectos. Vale a pena instalar este sistema, que para além de ser *open source*, ajuda no controlo e na análise das alterações de um projecto: possibilita ver que modificações foram feitas a dada revisão, ver diferenças entre diferentes projectos que estão num mesmo repositório, e descarregar o conteúdo do projecto num arquivo `.zip`. Para além de que é uma wiki com tudo o que de positivo existe no trabalho colaborativo numa wiki!

Previamente a instalar o Trac e associar a um projecto SVN é necessário ter instalado previamente o seguinte conjunto de *software*:

- Python [Fou09c], com uma versão superior ou igual à 2.3;
- Genshi [Sof09b], com uma versão superior ou igual à 0.5;
- Uma base de dados (SQLite [Hip09], PostgreSQL [Gro09] ou MySQL [AB09]) e os correspondentes *drivers* Python para a mesma.

Em Linux (distribuição Ubuntu [Can09]), para ter o Trac instalado basta fazer numa linha de comandos:

```
$ sudo apt-get install trac
```

Em alternativa, em Windows, é preciso descarregar o arquivo `.zip` ou o instalador `.exe` do Trac. Se se optar por descarregar o instalador `.exe`, é apenas necessário executá-lo e seguir as indicações que surgem no ecrã. Por outro lado, se se optar por descarregar o arquivo é necessário extraílo e seguir o passo (na pasta extraída):

```
python ./setup.py install
```

Este passo instala a ferramenta de administração trac-admin, usada para se criar e manter ambientes de projectos, bem como o servidor tracd.

Para se criar um ambiente para um projecto SVN previamente criado, em Windows ou em Linux, usa-se o comando trac-admin, como no se indica no exemplo seguinte para Linux (em Windows é semelhante mudando apenas o caminho para a pasta que vai ter o ambiente Trac):

```
$ trac-admin /home/projectos-trac/ProjX initenv
```

A pasta /home/projectos-trac/ProjX deverá estar previamente criada e vazia.

Durante este script será pedido para indicar a pasta onde se encontra o repositório SVN, o nome para o Projecto Trac, e mais informações (que podem ser deixadas com o valor por omissão, dando Enter para continuar a instalação do ambiente). Quando o script terminar, resta iniciar o servidor tracd e pondo-o a apontar para o projecto Trac já criado em /home/projectos-trac/ProjX:

```
$ tracd --port 8080 /home/projectos-trac/ProjX -d &
```

Desta forma, o servidor está à escuta na porta 8080. Quando abrirmos um browser e apontamos para o URL `http://localhost:8000/`, obtemos a página inicial da nossa wiki recém-criada, com o código do projecto SVN disponível após se pressionar o botão *Browse Source* (Figura B.2).

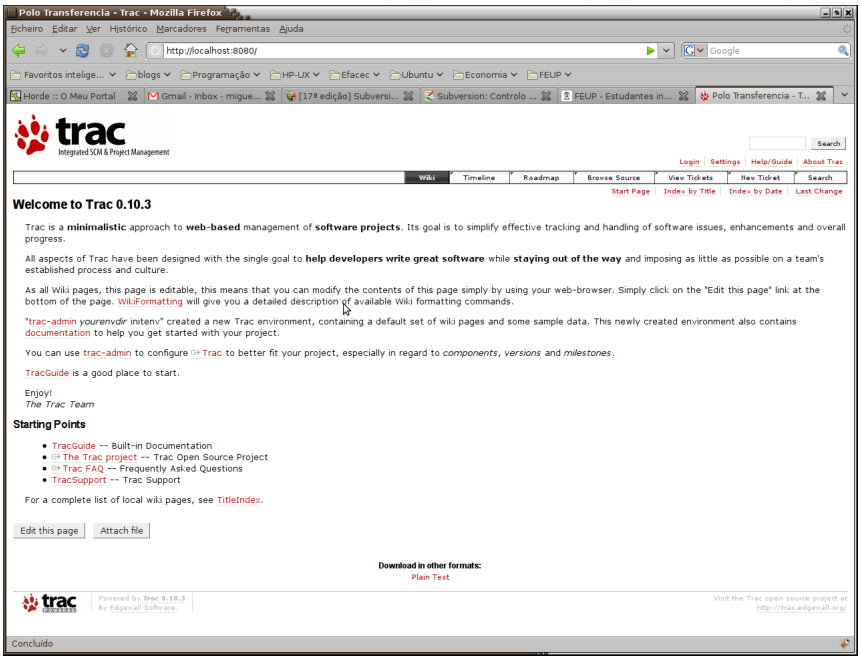

Figura B.2: Ambiente Trac num *web browser*.

Resta apenas começar a editar as páginas da wiki e a usar os plugins [Sof09d] disponíveis para usufruir de uma maior facilidade de trabalho colaborativo.

B.4 SVN no Eclipse (plugin Subclipse)

Nesta secção vamos ver como instalar o plugin Subclipse para o editor Eclipse.

Para adicionar suporte ao SVN no editor Eclipse (versão *Ganymede*), basta seguir os passos:

1. Aceder ao menu *Help* → *Software Updates...*
2. Aceder à *tab Available Software* e pressionar o botão *Add site...*
3. No URL escrever: `http://subclipse.tigris.org/update_1.4.x`
4. Pressionar o botão *OK*.

Uma nova entrada com o nome do URL vai surgir na lista de *software* disponível. Expandir a entrada e seleccionar o Subclipse e o adaptador JavaHL como se indica a seguir (Figura B.3).

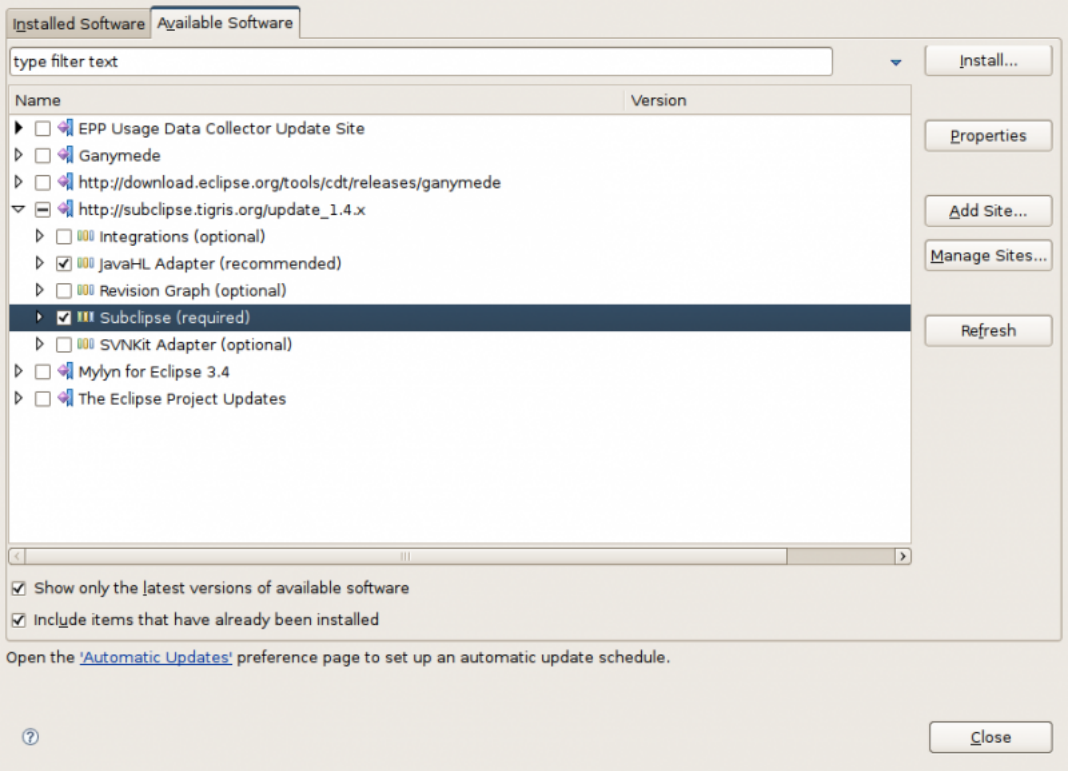

Figura B.3: Escolha do *plug-in* Subclipse para instalar no Eclipse.

Pressionar o botão *Install...* para que o Eclipse calcule as dependências deste *plug-in* e outros pacotes que sejam necessários descarregar. Na janela que surgir em seguida, pressionamos o botão *Next* >, aceitamos os termos da licença e pressionamos no botão *Finish* para dar início à instalação. Por fim, surgirá uma janela para reiniciarmos o Eclipse. Escolher *Yes* e temos neste momento o *plug-in* Eclipse correctamente instalado no ambiente de desenvolvimento Eclipse.

Podemos ver que temos o *plug-in* Subclipse ao aceder ao menu *File* → *New* → *Other...* e vemos a entrada SVN para fazermos *checkout* de um projecto SVN, como se indica na Figura B.4.

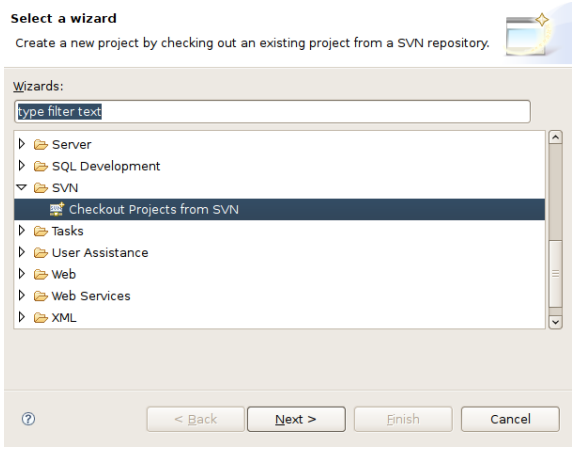

Figura B.4: Chekout de um projecto através do *plug-in* Subclipse.

Podemos ver na vista *Navigator* ou *Package Explorer* que o novo projecto contém os ficheiros que estão efectivamente no servidor SVN e que têm ícones diferenciados para indicar ao utilizador que se trata de um projecto SVN (ver Figura B.5) e do estado de cada um dos ficheiros que o compõem.

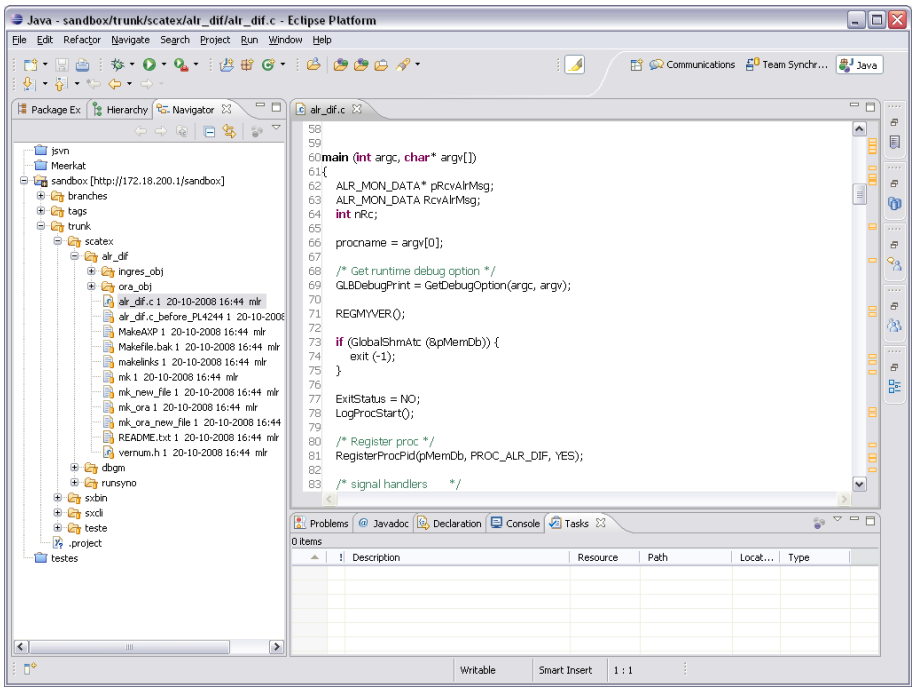

Figura B.5: Eclipse com o novo projecto SVN. Repare-se nos ícones que indicam o estado de cada um dos ficheiros.

Temos assim um ambiente de programação no IDE Eclipse totalmente integrado com o SVN.

Newton was a genius, but not because of the superior computational power of his brain. Newton's genius was, on the contrary, his ability to simplify, idealize, and streamline the world so that it became, in some measure, tractable to the brains of perfectly ordinary men.

G. M. Weinberg

C

Comparação entre Sistemas de Controlo de Versões

Neste anexo listam-se as principais características entre os SCM's apresentados na secção 2.5, separados pelo tipo: centralizado e distribuído.

SCM	Características
Perforce	Histórico de ficheiros e meta-dados; histórico de ficheiros renomeados, movidos, copiados, apagados e copiados de outros ramos; *merging* de 3 vias; Diferenças entre ficheiros de forma gráfica; *changelists*; commits atómicos; suporte para ASCII, Unicode, binário, *links* simbólicos e ficheiros UTF-16; suporte a RSS e notificações via e-mail
ClearCase	Controlo sobre os *builds*; possibilidade de não ter que repetir *builds* se dois objectos derivados têm a mesma configuração (isto é, os componentes de que dependem são apenas compilados uma vez e depois copiados entre diferentes objectos); integração com outros produtos da Rational, como o *ClearQuest* e o *RationalRose*, ou de terceiras partes, como o *Microsoft Visual Studio*, *Code::BLocks*, *NetBeans* ou *Eclipse*. Também existem plugins para *vim* e *Emacs*.
StarTeam	Suporta base de dados *Oracle* ou *Microsoft SQL Server* como base de dados dos repositórios; *branching* de 3 vias; gestão de controlos de acesso e gestão de segurança; *backups* "vivos"enquanto que o servidor está a correr; para além de fornecer todas as operações como SCM, guarda ainda os requisitos, as tarefas dos projectos, as alterações aos requisitos e as discussões internas entre *developers*.
Team System	Plataforma que agrega ferramentas para desenvolvimento integrado de *software* com ferramentas para percorrer todo o ciclo de desenvolvimento de *software*, melhorando a comunicação e a colaboração entre *developers*; contém ferramentas para desenvolvimento orientado a serviços (*Architecture Edition*), qualidade de software e performance (*Development Edition*), ferramentas de testes e de balanceamento de carga (*Test Edition*) ou de desenvolvimento de aplicações de base de dados (*Database Edition*).
SourceSafe	Fácil utilização e algum grau de integração com soluções de desenvolvimento da Microsoft; apenas recomendado para *developers* que queiram iniciar-se a aprender a usar um SCM; não permite versionar projectos grandes de software e não garante a integridade dos dados.
CVS	Permite acesso anónimo a um repositório (requer conta pessoal no servidor apenas para a operação de commit); customização de acções após um commit; possibilidade de *branching*, apesar de não ser tão fácil de utilizar como noutros sistemas (Subversion); usa compressão delta para salvaguarda eficiente das diferentes versões de um objecto versionado.
Subversion	Substituto natural do CVS, implementando novos *features* e corrigindo os problemas que o CVS ainda hoje apresenta; possibilidade de versionar directórios e de manter os dados e meta-dados (ao contrário do CVS); salvaguarda total de todo o histórico das alterações do *software*; commits atómicos, escolha de protocolos de rede de acesso (`HTTP`, `HTTPS`, `svn`, `svn+ssh`, `file://`); tratamento consistente dos dados, *branching* e *tagging* eficientes; API em C bem estruturada.

Tabela C.1: Comparação entre os SCM's centralizados.

SCM	Características
BitKeeper	Suporta cópias, renomeações, mover ficheiros ou pastas; commits atómicos; *changesets*; propagação de *changesets* e de *merges*; portável para UNIX, Linux e Windows.
Code Co-op	Desenvolvimento distribuído suportado por e-mail, LAN ou VPN; modelo baseado em *changesets*: modificações em vários ficheiros são colocados numa única transacção; operações de adição, renomeação, remoção de ficheiros são tratadas no mesmo nível como se fosse edições – podem ser adicionados em qualquer combinação num *check in* no *changeset*; *merges* de 3 vias; o histórico de um projecto é replicado em cada máquina, que pode também ser revisto, comparado e alterado; integração com produtos de terceiras partes, como o *Visual Studio*, *Visual Studio .NET*, *Borland Delphi*, *Borland C++ Builder*, *Starbase CodeWright* ou o *IBM Visual Age*
Git	Suporte para desenvolvimentos não lineares: suporta a construção rápida de ramos e de *merges*, e inclui ferramentas específicas para visualizar e navegar por um histórico não linear; cada *developer* fica com uma cópia local do histórico completo de desenvolvimento, e as alterações são copiadas do computador desse *developer* (que age como um repositório) para o de outro; os repositórios podem ser publicados via HTTP, FTP, rsync ou o protocolo git (via socket ou ssh); possui um servidor CVS para emulação, o que permite que clientes CVS possam usar plugins dentro de um IDE para aceder a repositórios Git; é muito rápido e escaláve; suporta autenticação criptográfica do histórico.
Bazaar	Grande performance, simplicidade e fácil de utilizar para *developers* acostumados a usar o CVS ou o SVN. É um híbrido, significando que é um SCM distribuído que pode funcionar com ou sem um repositório central; pode-se inclusivamente usar ambos os métodos ao mesmo tempo para um dado projecto; compatível com outros SCM's (como o SVN), o que permite aos *developers* criarem um ramo a partir de um sistema, realizar as alterações locais e em seguida um *commit* num ramo Bazaar, para mais tarde integrar as alterações no primeiro sistema.
Mercurial	É o resultado de um de dois projectos para SCM que foram iniciados para responder ao fim do suporte do BitKeeper a projectos *open source*; desenvolvido com o intuito de ser leve, fácil de usar e altamente escalável; apesar de ter um modelo conceptual muito distinto do CVS ou do SVN, a interface de linha de comandos é muito similar a estes, e assim sendo, torna-se muito intuitivo de usar, para quem está acostumado a usar o CVS ou o SVN; uma vez que foi desenvolvido para lidar com grandes repositórios, o Mercurial é em muitos casos, o SCM mais rápido disponível hoje em dia.

Tabela C.2: Comparação entre os SCM's distribuídos (1/2).

SCM	Características
Monotone	É outro SCM distribuído com uma filosofia muito diferente: os *changesets* são colocados num arquivo (especificamente, num arquivo `depot`), que vai coleccionando os *changesets* pretendidos a partir dos vários elementos da comunidade de *developers*; posteriormente, cada *developer* coloca o *changeset* pretendido no seu repositório privado; a segurança na comunicação dos dados é fornecida através de certificados RSA com um protocolo específico designado por `netsync`; identifica as versões dos ficheiros e directorias usando *checksums* SHA1; assim, pode identificar claramente quando um ficheiro é copiado ou movido, se a assinatura é idêntica e unifica as duas cópias; tem também um conjunto de comandos que tenta emular o CVS o mais possível, para facilidade de utilização deste SCM.
SVK	Escrito em Perl, com um *design* de hierarquias distribuídas comparável ao BitKeeper ou GNU Arch; usa o mesmo sistema interno de ficheiros que o SVN, mas distingue-se deste nos seguintes tópicos: operações como *check in*, *log* e *merge* são operações *offline*; os ramos são distribuídos; gestão leve das cópias de trabalho (não existem as directorias ocultas `.svn`, como no SVN); algoritmos avançados de *merge*, como o *star-merge* e *cherry picking*; *changesets* assinados e verificados digitalmente; pode ser usado com repositórios SVN, Perforce e CVS.
Darcs	É outra opção para um SCM distribuído, livre (licença GNU GPL), muito fácil de instalar e de montar um repositório para se começar a usar, e é baseado na filosofia de *patches* para distribuir alterações pela rede distribuída de *developers*, tal como no Git; o Darcs é escrito em Haskell e por isso tem de ser compilado com GHC. O GHC é um compilador de Haskell (ele próprio escrito em Haskell), com um procedimento de *bootstrapping* muito problemático, o que de certa forma poderá limitar o uso do Darcs em relação à escalabilidade deste SCM; outro problema neste SCM prende-se com o algoritmo de *merge* que, no pior caso, tem performance exponencial ao resolver os conflitos. Este problema foi minimizado na versão 2 do Darcs, mas ainda subsiste hoje em dia.
GNU Arch	Faz parte do projecto GNU e é licenciado sob a GNU GPL. Actualmente o GNU Arch está a ser mantido, mas não está a ser mais desenvolvido. O GNU Arch fornece: *Commits* atómicos (tal como no SVN); orientado a *changesets* para usar a filosofia de *patches*; rápidas ramificações: *branching* é eficiente e apenas anota a revisão anterior, pelo que o desenvolvimento continua a partir dessa revisão; *merging* avançado: devido à salvaguarda permanente de todas as revisões ancestrais e unificadas, o *merging* leva em conta que ramo contém determinado *patch* e assim fazer um *merging* de 3 vias baseado numa revisão ancestral; assinaturas Criptográficas; renomeação: todos os ficheiros e directorias podem ser facilmente renomeados; a uma operação de renomeação é atribuído um identificador único e não um nome, e uma vez que o histórico é preservado, os *patches* aos ficheiros são correctamente unificados mesmo se o nome dos ficheiros diferir nos ramos; controlo de meta-dados: as permissões e os *links* simbólicos são guardados e suportados da mesma forma que os habituais ficheiros e pastas.

Tabela C.3: Comparação entre os SCM's distribuídos (2/2).

On a visit to the NASA space center, President Kennedy spoke to a man sweeping up in one of the buildings. "What's your job here?"asked Kennedy. "Well Mr. President,"the janitor replied, "I'm helping to put a man on the moon".

Anónimo

D

Glossário de termos

Este anexo indica de forma tabular os termos mais usados no tópico de controlo de versões de *software*.

Baseline	Uma revisão de um artefacto a partir da qual se podem fazer alterações subsequentes.
Branch	Um conjunto de ficheiros versionados podem ser *ramificados* numa dada altura para que, a partir desse ponto no tempo, duas cópias desses ficheiros possam ser desenvolvidas em ritmos diferentes ou de formas diferentes por outros *developers*.
Alteração	Uma alteração (ou diff, ou delta) representa uma modificação específica sobre um documento versionado. A granularidade da modificação considerada varia entre sistemas de controlo de versões.
Change list	Em muitos VCS com *commits* atómicos, uma *change list*, *change set* ou *patch* identifica um conjunto de alterações feitas num só *commit*. Pode também representar uma vista sobre o código-fonte, permitindo examinar o código-fonte através do identificador único do *change list*.
Checkout	Um *check-out* (ou *checkout* ou *co*) cria uma cópia local de trabalho a partir do repositório, através de um número de revisão específico, ou do último existente (HEAD revision).
Commit	Um *commit* (*checkin*, *ci*, ou, mais raramente, *install*, *submit* ou ainda *record*) ocorre quando um conjunto de alterações feito na cópia local de trabalho é escrito ou unificado no repositório.

Tabela D.1: Termos mais utilizados no controlo de versões de *software* (1/2).

Conflito	Um conflito ocorre quando duas alterações são feitas por diferentes *developers* ao mesmo artefacto ou documento na mesma zona, e o sistema de controlo de versões é incapaz de unificar essas alterações sem ajuda. Um *developer* tem, neste caso, de resolver o conflito ao combinar as suas alterações com as do outro *developer* interveniente, ou ao seleccionar uma alteração em prole de outra.
Stream dinâmica	Uma *stream* (estrutura de dados que implementa uma configuração dos elementos num repositório) cuja configuração muda ao longo do tempo.
Export	Um *export* é similar a um *check out* com a excepção de que cria uma árvore de directórios limpa sem os meta-dados do versionamento usados na cópia local de trabalho. No caso do SVN, corresponde à árvore de directórios sem as pastas ocultas `.svn`.
Head	Corresponde ao *commit* mais recente.
Import	Um *import* é a acção de copiar uma árvore de directórios local para o repositório da primeira vez que se pretende popular um repositório vazio.
Label	O mesmo que *tag*.
Mainline	Similar ao *trunk*, mas pode existir uma *Mainline* por cada *branch*.
Merge	Um *merge* ou *integração* ou *unificação* é uma operação em que dois conjuntos de alterações são aplicados a um ficheiro ou conjunto de ficheiros.
Repositório	Um repositório é o local onde se encontram os ficheiros e o histórico das alterações, tipicamente num servidor. Em alguns casos, designa-se por *depot* (como no caso do SVK, ou Perforce).
Resolve	O acto de um *developer* intervir para resolver um conflito entre diferentes alterações no mesmo artefacto versionado.
Integração Reversa	O processo de *merge* de diferentes ramos no ramo principal do sistema de versões (*trunk*).
Revisão	Também designada por versão: é qualquer tipo de alteração.
Tag	Uma *tag* ou *etiqueta* refere-se a um *snapshot* no tempo, consistente ao longo de vários ficheiros. Estes ficheiros neste ponto temporal podem ser *etiquetados* com um nome facilmente perceptível ou um número de revisão. Por exemplo, à revisão 123 corresponde a *tag* "Versão 2.0.1 com o *bug* Alpha corrigido".
Trunk	A linha principal de desenvolvimento.
Update	Um *update* (ou *sync*) unifica alterações que foram feitas no repositório (por outros *developers*) na cópia local de trabalho.
Cópia de trabalho	A cópia de trabalho é a cópia local de ficheiros a partir do repositório, num momento específico ou revisão. Todo o trabalho existente no repositório é inicialmente produzido na cópia de trabalho e posteriormente colocado no repositório (através do *commit*).

Tabela D.2: Termos mais utilizados no controlo de versões de *software* (2/2).